SIEMENS

Teamcenter Integration for Creo® 2412

Administration Guide

SIEMENS AND ITI PROVIDE THIS PROGRAM AS IS AND WITH ALL FAULTS. SIEMENS AND ITI SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR USE. SIEMENS AND ITI DO NOT WARRANT THAT THE OPERATION OF THE PROGRAM WILL BE UNINTERRUPTED OR ERROR FREE.

Siemens and the Siemens logo are registered trademarks of Siemens AG. Teamcenter is a trademark or registered trademark of Siemens PLM. This software and related documentation are proprietary to Siemens PLM. ©2024 Siemens AG. All rights reserved.

Creo® is a registered trademark of Parametric Technology Corporation.

Copyright© 2024 International TechneGroup Incorporated, 5303 DuPont Circle, Milford, Ohio 45150, U.S.A. All Rights Reserved.

This program contains software licensed from Siemens PLM. ©2024 Siemens PLM. All Rights Reserved.

Contents

Chapter 1 Introduction 1-1

Chapter 2 Configuration 2-1

ipem.properties 2-1
- General preferences 2-2
- Open preferences 2-4
- Save preferences 2-6
- Active Workspace preferences 2-7
- Cache Manager preferences 2-7
- MultiCAD preferences 2-8
  - Example 1 – Direct reuse 2-8
  - Example 2 – Indirect reuse 2-9

ipem.xml 2-10
- Attribute Mapping 2-10
  - cad_name 2-12
  - pdm_name 2-16
  - default 2-18
  - missing_attribute_action 2-18
  - direction 2-19
  - constraint 2-20
  - truncate 2-21
  - Attribute Mapping and Family Tables 2-21
  - Loading the Attribute and Column Mappings into Teamcenter 2-26
- Properties display in the Save and Teamcenter New Dialogs 2-28
- Configuring the Properties Panel for Teamcenter New and Save Operations 2-28
  - Required properties 2-29
  - Optional properties 2-31
- Property naming rules display in the Save and Teamcenter New Dialogs 2-33
- Configuring the ID Generator Dialog 2-34
- Properties enabled in Save Dialog for Second Save 2-35
- User Defined Columns 2-36
  - Column Map 2-38
- BOM Map 2-40
  - BOM View Type 2-40
  - BOM mapping for Part and Assembly MultiCAD reuse 2-40
  - BOM mapping for Part-only MultiCAD reuse 2-41
  - BOM Properties 2-42

Configuring BOM Property Mappings 2-43
Mappings to types other than Feature Parameters 2-44
Auxiliary Files 2-45
General Auxiliary Files 2-45
Specialized auxiliary files 2-51
General Substitution Keywords 2-62
Template Map 2-63
Using Templates from a Directory 2-64
Using Templates from Teamcenter 2-65
Dataset Map 2-65
Allowed Item Types 2-66
Submit to Workflow 2-67
Managing ipem.properties and ipem.xml in Teamcenter 2-67
Teamcenter Weight and Balance Support 2-70
Teamcenter Preferences 2-70
PTC configuration settings 2-71
config.pro 2-71
config.sup 2-75
config_init.mc 2-75
JT translation preferences 2-76
Teamcenter DisplayName Constant 2-76
Multi-Site 2-77
Configuring Teamcenter Multi-Site to Export Creo® Models 2-77
Configuring Teamcenter Multi-Site for Transferring Ownership of Creo® Models 2-78
Configuring replica icon in Integration dialogs 2-79

Chapter 3 3-80

Chapter 4 Utility Functions 4-1

Exporting Creo® Models from Teamcenter with ipemexport 4-1
Usage 4-1
Description 4-2
Family Tables 4-6
Multiple Model Versions 4-6
Instruction Files 4-6
Examples 4-8
Importing Creo® Models into Teamcenter with ipemimport 4-9
Usage 4-9
Description 4-9
Item IDs, Names, Types, and Revisions 4-13
Attributes 4-14
BOMs 4-14
Family Tables 4-14
UDF Files 4-14
Multiple Model Versions 4-15
Auxiliary Files 4-15
Instruction Files 4-15

Examples 4-18
Renaming Creo® Models in Teamcenter 4-19
Renaming with ipemrename 4-19
Usage 4-19
Description 4-20
Suppressing the GUI 4-22
Examples 4-23
Renaming Creo® Models in Teamcenter with Creo®→Rename 4-24
Upgrading the Teamcenter Database 4-26
Usage 4-26
Fixing missing deferred rename data 4-28
Upgrade family tables for Multi-site 4-29

Chapter 5 Data Model 5-30

Integration Relations loaded by the BMIDE Template 5-30

Chapter 6 Upgrading Creo Versions 6-1

When to reinstall the integration 6-1

Chapter 7 Troubleshooting 7-2

Reporting problems 7-2
Failure to start or connect to Creo® 7-3
Command line utilities 7-3
From Teamcenter 7-4
Starting Creo® from Teamcenter 7-4
Exporting data from Teamcenter 7-4
Failure to load the Teamcenter Integration for Creo® 7-5
Cache Manager launch issues 7-11
Troubleshooting runtime issues 7-11
Send to Creo® from Teamcenter Rich Client does not launch 7-13
Send to Creo® from Active Workspace does not work 7-13
Renaming Models Imported from Pro/PDM® 7-13
BOM Synchronization 7-14
Opening seed parts from Teamcenter 7-14
Opening pre Creo® 4.0 simplified representations from Teamcenter 7-15
Teamcenter operations on Creo® data 7-15
Failure when saving or importing Creo® family tables 7-17
Creo membership relation not created 7-18
Failure to duplicate Creo® data 7-18
Proper Installation 7-18
Notification 7-18
Common Problems 7-19
Performance troubleshooting 7-20
Deploy performance journaling on the integration client 7-20
Test case execution 7-20

Chapter 8 Best Practice Recommendations 8-1

Properties display in the Save As, Save All, Save Checkouts and Teamcenter New Dialogs..... 8-1
Configuring Teamcenter for Single Workflow 8-2
Restricted access to data..... 8-3
    Opening a model with access restrictions 8-3
Creo® Skeleton Models..... 8-4
MutliField Keys 8-4
Format Files..... 8-4

Chapter 9 Java Customization points..... 9-5

Available User Exits..... 9-5
Integration generated XML 9-6
    Tags..... 9-6
        ipem_user_exit..... 9-6
        auxiliary_files..... 9-7
        cadtopdm_control..... 9-7
        file_name 9-7
        item_id 9-7
        item_name 9-7
        item_revision_id 9-7
        item_type 9-8
        model..... 9-8
        properties..... 9-8
        pdm_property..... 9-8
        value 9-8
    Sample XML generated by the Integration 9-9
Response XML from custom code 9-15
    Response Tags..... 9-15
        save_operation_response 9-15
        action 9-15
        model..... 9-15
        properties..... 9-15
        pdm_property..... 9-16
    Sample Response XML 9-16

Chapter 10 VC++ Customization points..... 10-18

Available User Exits..... 10-18
    AuxiliaryFileUserExit()..... 10-18
        Example XML..... 10-19
Modifying the User Exit DLL..... 10-20
Building and Installing Production IpemUserExit.dll 10-21
Debugging IpemUserExit.dll..... 10-22

Chapter 11 Appendix..... 11-25

Auxiliary files..... 11-25

Specialized Auxiliary Files 11-25
JT Files 11-25
create_text_file 11-30

Chapter 1 Introduction

When first installed, the Teamcenter® Integration for Creo® is configured to satisfy most customers' requirements. As users become more experienced with the product, the Teamcenter Administrator may want to change certain settings to align with customer-specific requirements and processes. The ***Configuration*** section of this document explains how to change the configuration, and why you may want to do so.

The integration provides a set of utility programs to perform administrative-level functions on Creo® data. Those functions, and their usage, are described in the ***Utility Functions*** section of this document.

Finally, there is a small set of known issues that customers may encounter when working with the product. These are covered in the ***Troubleshooting*** section.

For technical support, please contact the Siemens Support Center. You can reach Support Center via the World Wide Web at

https://support.sw.siemens.com/en-US/

Or by telephone:

United States and Canada (800) 955-0000

In other countries, please see

http://support.industrysoftware.automation.siemens.com/services/global_number.pdf

For additional telephone numbers, or call (714) 952-5444.

Chapter 2 Configuration

The integration's CAD data management behavior is controlled by two files:

- ipem.properties
    - A text file containing a list of keywords and values. Each keyword is documented within the file, and each has a default value. Together they form a set of preferences which are read when Creo® is launched with the integration loaded. If the value for a keyword is not modified, then the default value remains in effect. Some preferences in the ipem.properties file are best controlled at the site level, while others may be user specific. The Teamcenter Administrator at each site must decide which settings users are allowed to override.
- ipem.xml
    - A set of maps to control integration behavior at the site level. This file is not designed for modification by anyone but the Teamcenter Administrator.

JT translation, when performed on the integration clients during save to Teamcenter, relies upon a set of template files in the pejt subdirectory, from which the runtime tessellation configuration files are derived. These template files may be modified as needed if certain system-level settings remain unchanged.

While not directly read by the Teamcenter Integration for Creo®, the Creo® session preferences can influence integration behavior. These preferences are typically configured in the PTC config.pro file, which may be found in any of several locations.

Similarly, a small set of Teamcenter preferences either are read directly by the integration or have some bearing on how Creo® data is managed in Teamcenter.

ipem.properties

The `ipem.properties` file, located in the client installation directory, contains the preferences that can be configured for a site or for an individual user.

The `ipem.properties` file may be placed in any of these locations:

- Teamcenter, as described in Managing ipem.properties and ipem.xml in Teamcenter. If the **ipem.properties** file is managed in this way, then the remaining locations in this list are ignored.
- In the working directory where the Teamcenter Integration for Creo® is started.
- In the user's home directory.
- In the Teamcenter Integration for Creo® installation directory.

When the Integration starts, it reads all `ipem.properties` and `.ipemrc` files in these locations. If several files specify the same preference, the preferences from the working directory take precedence over those from the user's home directory, and preferences from the user's home directory take precedence over those from the installation directory. Preferences that are likely to have the same definition for all users, such as should be specified in the installation directory, while a user's personal customizations may be specified in the home directory or working directory preferences file.

It is recommended that the Teamcenter administrator maintain the installed ipem.properties file as a read-only master copy, and make all changes in a separate file, which is included in ipem.properties using the preferences.file setting, as in this example:

```
preferences.file = .\\ipem_site.properties
```

This technique allows you to override each preference's default value, while preserving the original unchanged value for documentation. It also helps to quickly identify those preferences whose values have been changed by the site.

Special syntax is available for "locking" settings so that users cannot override them, and for resolving values through environment variables (especially useful for settings that point to directory or file locations). This syntax is explained at the top of the ipem.properties file, and so it is not repeated here.

Any of the published configuration settings may be overridden in the ipem.properties file, but some are more frequently changed than others. The following is a summary of the more commonly modified settings.

Note

If the ipem.properties file is managed as a dataset in Teamcenter, as described in Managing ipem.properties and ipem.xml in Teamcenter, all local copies are ignored.

General preferences

Preference	Description
table.columns.std:	With the default value, the integration displays the most commonly needed attributes of the CAD models. It is possible to display additional or alternative attributes, either by selecting from a predefined set (documented in `ipem.properties`) or by configuring display of custom columns. The attributes included in `table.columns.std` apply to all integration dialogs, unless overridden by similar dialog-specific preferences, such as `table.columns.checkin` and `table.columns.update`.
log.file	The integration automatically writes a log file to the session's startup directory, the location of which may be influenced by several factors including the configuration of desktop shortcuts. Assuming that all users have write access to a consistent location on their client workstations, it is recommended to set log.file to a fixed value for all users, such as `c:\\temp\\txdlog.txt`.

log.level	The default setting for this option is `INFO`, which includes the basic information needed for troubleshooting. Whenever reporting an issue to Support Center, it is important to increase the level of detail in the log file. This may be done by setting `log.level` to a value of `DEBUG`. The most complicated cases will require a value of `TRACE`.
summary.table.rows.std	This preference controls the attributes which are displayed in the Summary panel on the **Open**, **Open Dependencies**, and **Cache Manager** dialogs. The `ipem.properties` file documents a predefined list of keywords that may be used to add or change the set of displayed attributes. In addition, any Teamcenter attribute (out of the box or custom) may be added to the set by first configuring the attribute as a mapped column (see the section on column mapping, later in this document) and then adding the column map's `display_name` value to `summary.table.rows.std`.
registernew.enable	This preference controls the display of the **Teamcenter New** dialog when a new model is created in Creo®. With the default value of `prompt`, the integration will ask the user whether he wants to manage the new model in Teamcenter.
pe.checkout.reminder	The Integration encourages users to check out models before modifying them, to prevent accidental overwrites and lost productivity. Most Creo® edit functions are intercepted and the user is prompted to check the model out. The prompting can be suppressed by changing this preference to `false`.
pe.checkout.reminder.commands.add	This preference is a colon separated list of Creo operations to add to the internal list used by the integration's **Checkout Reminder** feature. Adding an operation to this preference causes the checkout reminder dialog to appear when Creo triggers the operation. Operation names can be identified by examining the Creo trail file as actions are performed, then matching a specific operation of interest to the corresponding **~ Command** entry.

pe.checkout.reminder.commands.remove	This preference is a colon separated list of Creo operations to remove from the internal list used by the integration's **Checkout Reminder** feature. Adding an operation to this preference prevents the checkout reminder dialog from appearing when Creo triggers the operation. Operation names can be identified by examining the Creo trail file as actions are performed, then matching a specific operation of interest to the corresponding **~ Command** entry..

Open preferences

Preference	Description
checkout.deferredrename.enable	Change this preference from **true** to **false** if you first deployed integration version 11.3.0 or later.
checkout.dic.nx.default.jt.name	Only applicable when NX CAD models are loaded into Creo® for use as reference geometry, using the integration's Design in Context feature. The NX administrator may have changed the default reference set to something other than `MODEL$`, resulting in JT datasets having more than one JT named reference. This preference must be set to the named reference name that contains the desired JT geometry, which is typically the Brep JT.
filters.checkout.choices	Filters are check boxes that appear on the integration's **Open** dialog. They give the Creo® designer control over how much CAD data is downloaded from Teamcenter to his client. A subset of the available filters is displayed by default. Filter options can be added or subtracted by modifying this preference.
filters.checkout.default	Once the available filters are configured, it is advisable to set them to the appropriate default true or false values, so that users do not have to change them frequently. Filter options that are included in the `filters.checkout.default` preference are set to `true` by default; filters not mentioned in this preference default to `false`.
checkout.dependencies.default	This preference controls the default value of the **Show Dependencies** checkbox on the **Open** dialog. If `true`, the **Show Dependencies** checkbox is selected when the **Open** dialog appears, and the Cache Manager's

	Update operation will display the **Dependencies** dialog when additional dependencies are being retrieved. If `false`, the checkbox will be unselected on the **Open** dialog and the **Dependencies** dialog will not appear during the Cache Manager's **Update** operation. A value of `true` is the default.
checkout.autosave.attributes	With the OOTB 'true' value, the integration will automatically save to disk any model that is updated by pdmtocad attribute mapping. On some systems the Creo® save to disk can be very slow and changing this preference to 'false' prevents the integration from performing the save. The parameter values will still be updated in the Creo® session, and users can explicitly save changed models to disk, as needed, to preserve the values that were mapped from Teamcenter.
iman.bomrevisionruledefault	When this preference remains unset, the default revision rule that is applied to the **Open** operation is determined by the user's Default Revision Rule setting, in the Rich Client's **Options | Product Structure** dialog. Customers may want to configure a different revision rule as the default for Creo® users, and this can be done by setting `iman.bomrevisionruledefault` to the name of any revision rule defined in Teamcenter.
iman.preciseisimprecise.familytable	This preference controls how the relationship between a family table instance and its generic model is traversed during the **Open** operation. With the default value of `false`, the precise (last-saved) revision of the generic will be downloaded for each instance. When changed to `true`, the generic's revision will be resolved using the designated revision rule. Customers planning to revise generics without revising all instances should retain the default value of `false` for this preference.
checkout.bom.attributemapping.onelevelonly	When a site has configured the integration to map properties from the Bill of Materials (BOM) to Creo® feature parameters in the assembly, this preference controls whether such mapping is done at all levels, or only the top level, of the assembly. For performance reasons, it is recommended that customers retain the default 'true' value unless they have a specific reason for changing it to false.

Save preferences

Preference	Description
checkin.after	This setting controls the default value of the **Work Folder After Save** control on the save dialog. Some customers may wish to change this default so that users do not have to remember to do it themselves, during each save operation.
checkin.arrangements	This setting is `false` by default, due to the performance impact of creating and updating arrangements in Teamcenter. Changing this preference to `true` will cause the integration to save the parts list for each user-defined simplified representation as an arrangement in Teamcenter, which may then be used in the embedded viewer, or to optimize the performance of a subsequent load operation. The Teamcenter Administrator at each site should evaluate whether changing this preference to `true` results in a net performance gain or loss.
checkin.owner.integration.ignore	This preference controls whether models can be saved to existing Teamcenter Item Revisions that could be owned by another integration. Ownership of an Item Revision is determined by the existence of an integration-specific dataset. For the Creo® integration, those datasets are Pro* (ProAsm, ProPrt, ProDrw …). The default setting of `false` prevents saving models to existing Item Revisions that do not already contain a Creo® dataset. A value of `true` allows saving models to all Item Revisions - even those potentially owned by other integrations. It is safest to retain the default setting of `false`.
checkin.saverelated.drawings.enabled	When modified parts and assemblies are saved to Teamcenter, the integration can look for related drawings in the current workspace, and if found, will save them to Teamcenter. This preference determines whether the integration searches for related drawings while saving modified parts or assemblies. The default value is **true**.
checkin.saverelated.drawings.rebuild	By default, all drawings added to the **Save** dialog are rebuilt. The rebuilding does take time. Change this preferences's value to either **prompt** or **never**, if you decide that automatic rebuilding of drawings is not needed or desired.
eai.jtenabled	When the ProeToJT translator is installed, the integration can call the translator to create JT files during save to Teamcenter, and then automatically upload the resulting JT files into the Teamcenter Bill of Materials. The translation is controlled by a check box on the save dialog, labeled **Save JT Files**. Because client-side JT translation can have a

	significant impact on save performance, Administrators may wish to disable this function by default, and this is done by retaining the default value of `false`.
iman.addonlynewtoselected	By default, all new items are put into a Teamcenter folder. If the user selects a folder in the save dialog, then the items are put into that folder. If he does not select a folder, then the new items are put into the Newstuff folder. Setting this preference to `false` tells the integration that not only new items, but also existing items being saved as new versions or new revisions, should be put into the destination folder. This creates links in Teamcenter such that a given item may belong to multiple folders. A typical use case for this is to collect all the components for a large assembly structure into a single folder, so that Teamcenter users can easily find them together.
checkin.ignoremissing	Customers often have CAD assemblies that contain references to other models that are no longer used by the assembly. These are sometimes called “ghost references”. When an assembly is saved to Teamcenter, the integration will find and report these as missing references and will ask the user whether to continue with the save. Another scenario in which the warning dialog appears is when the assembly has suppressed components that have not already been saved to Teamcenter. Some customers have so much data with this condition that they prefer not to see the warning dialog on each save, and this can be configured by changing the preference from `prompt` to `always`.
pe.modelcheck.output	When `pe.modelcheck.show.report` is `true`, then this preference must be set to a valid directory location on disk, and the user must have write permission to that directory.

Active Workspace preferences

Preference	Description
activeworkspace.enabled	This preference controls whether the **Active Workspace** tab appears in the Creo® Navigation panel. This should only be set to `true` if Active Workspace is installed on the Teamcenter server.

Cache Manager preferences

Preference	Description

Pro2_CacheFolder	All user workspaces are created and managed under the specified directory location. The OOTB value of this preference is %USERPROFILE%\Siemens\creocache.
iman.workspace.exception.folders	The integration prevents users from saving Creo® files that reference Creo® files from locations other than a managed workspace. In some cases, files such as feature libraries will be accessed from an outside location, and this preference identifies such locations. To manage the external files to Teamcenter, the recommended is that an admin save the files and restrict the users from modifying. Users are expected to use the files in their design, but not to modify them. All files in all subdirectories below the specified locations are eligible for saving to Teamcenter. DOS 8.3 short paths are not supported; you must use the full path(s), with double backslash (\\) delimiters. UNC paths are not supported.
iman.workspace.maxnumberfiles	It is important, for optimal performance, to prevent any single managed workspace from growing too large in terms of its file count. An excessively large number of files in a workspace will degrade the performance of Cache Manager launch and refresh. This preference allows site administrators to constrain workspaces to a size that will accommodate the largest assembly expected to be saved to, or opened from, Teamcenter.
table.columns.update	**Cache manager** property columns can be added to, subtracted from, or reordered with this preference. Only Teamcenter properties may be included; Creo® model parameters are not eligible. The list of OOTB property columns is described in the **ipem.properties** file. Additional custom properties can be displayed as columns, when configured with the **column_map** in **ipem.xml**. Only custom columns defined with the **pdm_name** tag may be used in the **table.columns.update** preference.

MultiCAD preferences

Users can incorporate designs from other Mechanical CAD systems into their Creo® designs, leveraging the design reuse capabilities of Creo® itself. In some cases, the non-native models can be directly reused, without first converting them to a neutral format. In other scenarios, direct reuse may not be desired or even possible, and the non-native design is first converted to a neutral format, preferably JT, and then opened or inserted into a Creo® assembly. Consider the following examples:

Example 1 – Direct reuse

A customer using Creo® 7.0.2.0 wishes to reuse a SOLIDWORKS® design as a subassembly. His SOLIDWORKS® design was last saved to Teamcenter with SOLIDWORKS® 2019. Creo® 7.0.0.0 can directly open SOLIDWORKS® 2019 and earlier files, so direct reuse of the SOLIDWORKS® design data is supported.

Note

Direct reuse enables assembly-level as well as part-level MultiCAD use cases, because the non-native assembly, when opened or inserted into Creo®, recognizes and loads its own child components. The integration has only to download the files and make them available to Creo®, and Creo® does the rest of the work.

Each Creo® version supports specific versions of the other Mechanical CAD tools. Before configuring the integration to directly reuse non-native MCAD data, test or contact PTC to ensure the CAD versions are compatible. You can also consult the UNITE support matrix here: https://www.ptc.com/en/support/article/CS134024

Direct MultiCAD reuse can only be supported with Creo® 6.0.4.0 and higher. Earlier Creo® versions were not tested and cannot be supported.

Additional configuration steps involving the preview_command in the auxiliary_file_map are required to see SOLIDWORKS® previews in the Creo® integration dialogs.

Example 2 – Indirect reuse

The same customer upgrades SOLIDWORKS® to version 2021 but does not upgrade Creo®. To open SOLIDWORKS® 2021 files, a minimum of Creo® 7.0.4.0 is required, therefore direct reuse cannot be supported. However, Creo® 7.0.2.0 can open JT version 10.2 and earlier, and the SOLIDWORKS® 2021 JT translator writes version 10 JT files, so **indirect** reuse, via JT, is supported.

Because under Teamcenter management, assembly structure is not stored in JT files, the integration recognizes only the part JT files as reusable, and builds the non-native assemblies on the fly, each time they are opened or inserted from Teamcenter. The dynamically constructed Creo® assemblies are transient, and they cannot be saved to Teamcenter.

This approach is also applied to NX assemblies, as a special case, because NX assemblies, when managed in Teamcenter, can only be directly opened by the NX integration. See the explanation of the **assembly="ignore"** tag, in the bom_map section of this document, for more details.

When indirect reuse is configured for any non-native CAD type, be sure to include a **template_map** for the **ProAsm** dataset type in the **ipem.xml** file, as described in this section: Using Templates from a Directory. The integration needs the template file to build the transient placeholder Creo® assemblies.

The following preference determines which file types are recognized as eligible for direct MultiCAD use cases.

Pro2_Foreign_Datasets

This preference is a colon-separated list of 3D foreign datasets that the Creo® integration recognizes. Possible values are:

- SWAsm SOLIDWORKS® assembly
- SWPrt SOLIDWORKS® part
- AIAssembly AutoDesk Inventor® assembly
- AIPart AutoDesk Inventor® part
- SE\u0020Assembly SolidEdge® assembly
- SE\u0020Part SolidEdge® part
- CATPart Catia® part
- CATProduct Catia® assembly
- UGMASTER NX® part

Only include the dataset types that are useful in collaboration at your site. The ordering of the list determines the order in which the integration searches for datasets. For example, if you prefer to reuse a SOLIDWORKS® part dataset, and to revert to an NX® dataset if a SOLIDWORKS® part is not available, then ensure that **SWPrt** appears before **UGMASTER** in the colon-separated list.

Note that the **DirectModel** (**JT**) dataset type is not a valid value for this preference. That is because indirect reuse, via JT, is automatically triggered when direct reuse has not been configured via the **Pro2_Foreign_Datasets** preference. In other words, if the integration has not been explicitly told to directly reuse the non-native part or assembly, and JT files are available, they will be found and used.

ipem.xml

The `ipem.xml` file, located in the client and server installation directories, tells the Integration how to map Creo® parameters to Teamcenter attributes, which BOMs to create when a model is saved to Teamcenter, whether auxiliary files should be saved with models, which user-defined columns to display, and which Teamcenter dataset and named reference types to use for Creo® models.

The server copy of `ipem.xml` is used only as a source for attribute mapping definitions, which are then loaded into the Teamcenter database. The copy of `ipem.xml` on each integration client controls all runtime behavior at the site level. The server and client copies of `ipem.xml` should be identical.

Attribute Mapping

When a model is exported from Teamcenter and opened in Creo®, properties assigned to the item, item revision, dataset, or forms may be copied to parameters in the Creo® model. When a model is saved from Creo® to Teamcenter, its parameter values may be copied back to these objects. The attribute map specifies which properties should be copied for a particular type of Creo® model. Some properties appear in the properties panel of the **Save** dialogs and can be modified. The display of these properties is controlled by **Operation Descriptors** in Teamcenter. Additional properties can be added to the properties panel by including them in the **Operation Descriptors**. Learn more by reviewing **Properties display in the Save and Teamcenter New Dialogs** elsewhere in this document.

The **ipem.xml** file in a new installation contains sample attribute mapping definitions. It is recommended that new customers start with the sample mapping and then adjust as needed. The mapping must be uncommented in **ipem.xml** and then loaded into Teamcenter, for it to take effect at runtime. The following is a copy of the sample `attribute_map` definitions.

```
<attribute_map cad_type="ASM:PRT">
    <attribute>
        <cad_name value="Description"/>
        <pdm_name value="item_name"/>
        <direction value="cadtopdm"/>
    </attribute>
    <attribute>
        <cad_name value="Description"/>
        <pdm_name value="ItemRevision.object_name"/>
        <direction value="both"/>
    </attribute>
    <attribute>
        <cad_name value="Revision"/>
        <pdm_name value="item_revision_id"/>
        <direction value="pdmtocad"/>
    </attribute>
</attribute_map>
```

Within an `attribute_map` definition can appear zero or more `attribute` definitions. Each attribute definition specifies the name of a parameter in the Creo® model (this is the attribute's *CAD name*), the name of the corresponding property in Teamcenter (its *PDM name*) and additional information describing its default value, what to do if the CAD parameter does not exist, and so on. The property's CAD name and PDM name must be specified, and they must be the first and second tags, respectively, in an `attribute` definition. The other elements of an attribute definition are optional and may appear in any order.

For example, here is an attribute map for Creo® assemblies and parts:

```
<attribute_map cad_type="ASM:PRT">
  <attribute>
     <cad_name value="PartNumber"/>
     <pdm_name value="item_id"/>
     <missing_attribute_action value="create"/>
     <direction value="pdmtocad"/>
  </attribute>
  <attribute>
     <cad_name value="Weight"/>
     <pdm_name value="ItemRevision Master.wgt"/>
     <default type="Double">1.0</default>
     <direction value="cadtopdm"/>
  </attribute>
</attribute_map>
```

The `cad_type` defines the type of Creo® model to which this map applies. The allowed `cad_type` values are the three-letter extensions that Creo® uses to identify its model files, in this case "ASM" for assemblies and "PRT" for part models. The `cad_type` value is case sensitive and must be capitalized. If more than one type is specified, use a colon to separate the type names.

The example above specifies only two attributes to be mapped between Teamcenter and Creo®. One is the part number, which the map says is in a parameter named `PartNumber` in Creo® assembly and part models and is the item ID in Teamcenter items. The other is the weight of the assembly or part, which is in a Creo® parameter named `Weight` and in a property named `wgt` in the item revision master form. The part number is copied only from Teamcenter to Creo® (the direction is `pdmtocad`), while the weight is copied only from Creo® to Teamcenter (the direction is `cadtopdm`). A parameter for the part number will be created if it does not exist in the Creo® model (the `missing_attribute_action` is

create). If the weight does not exist in the Creo® model, a default value of 1.0 will be stored in the form property (this is specified by the default tag).

Each attribute tag is a single mapping of Creo® parameter and Teamcenter property. A parameter can be mapped to multiple properties by adding additional attribute tags for each mapping. Similarly, a property can be mapped to multiple parameters by adding additional attribute tags. The integration does not validate multiple mappings; each mapping is performed without respect to any rules. It is a site administrator's responsibility to establish a useful set of mappings.

If a model contains a family table, the PartNumber parameter in this example should be a column in the family table; otherwise, every family table member will share the same parameter value and will appear to have the same item ID. See the section on the constraint tag for more information. The tags that can be used in an attribute definition are described below. Some of these descriptions refer to *source* and *destination* attributes. When opening a model, the source properties are those in Teamcenter and the destination attributes are the parameters in Creo®. When saving a model, the source attributes are the parameters in Creo® and the destination properties are those in Teamcenter.

Note

Ambiguous attribute mappings must be avoided to ensure consistent results. For instance, do not configure both parts and drawings to write to the same property in Teamcenter. In such a case, if the property value in a drawing differs from that of its part, the last model saved will determine the value of the Teamcenter property. Do not map two different Teamcenter properties to the same property in Creo®. If the Teamcenter properties have different values, the value in Creo® may change unexpectedly.

Date mappings should always go to a string in the target. Do not map dates with a direction of both.

Date mappings in cadtopdm direction, will not clear Teamcenter value when CAD value is set to blank.

cad_name

```
<cad_name value="name"/>
```

The cad_name tag specifies the attribute's name in the Creo® model. This may be the name of a model parameter in Creo®, or it may be the name of an attribute function. If it is the name of a parameter, it must obey the usual rules for Creo® parameter names, such as using only alphanumeric characters and the _ character. The cad_name tag is required, and it must be the first tag in an attribute definition. It has no default.

The following attribute functions may be used instead of a parameter name, but only when direction is "cadtopdm".

has_alf()	The attribute value is true if the Creo® model has an assembly level feature.
is_bulk()	The attribute value is true if the Creo® model is a bulk item part. The value is false if the model is not a bulk item part.
is_ext_simp_rep()	The attribute value is true if the Creo® model is an external simplified representation. The value is false if the model is not an external simplified representation.
is_family_member()	The attribute value is true if the Creo® model is a family table generic or instance. The value is false if the model is not a generic or instance.

is_generic()	The attribute value is true if the Creo® model is a family table generic. The value is false if the model is not a generic.
is_instance()	The attribute value is true if the Creo® model is a family table instance. The value is false if the model is not an instance.
is_interchange()	The attribute value is true if the Creo® model is an interchange assembly. The value is false if the model is not an interchange assembly.
is_skeleton()	The attribute value is true if the Creo® model is a skeleton part. The value is false if the model is not a skeleton part.
is_top_generic()	The attribute value is true if the Creo® model is a family table generic and not a family table instance. The value is false if the model is not a generic or if the model is a generic that is also an instance.
version_stamp()	The attribute value is the Creo® model's version stamp.

The following parameters have special meaning to PTC and have limited support through the Teamcenter integration.

PRO_MP_MASS

The integration can read the value for a cadtopdm mapping, provided the parameter has a value. If a value does not exist, a PTC exception is likely to occur, causing the save operation to fail. To avoid this problem, customers should map another parameter that contains the value of PRO_MP_MASS.

- In ipem.xml map a Creo® model parameter IPEM_MASS to the desired property in Teamcenter and load the mappings into Teamcenter.
- In the Creo® models, define a parameter named IPEM_MASS, of type Double
- In Creo®, enable mass property calculation by setting the config.pro property mass_property_calculate to automatic
- In Creo® create a relation between the IPEM_MASS parameter and the system-defined PRO_MP_MASS parameter (Tools→Relations→IPEM_MASS=PRO_MP_MASS). The creation of this relation will require that the PRO_MP_MASS parameter has a value (generally set by a regeneration)

When saving a model to Teamcenter, IPEM_MASS will be saved to Teamcenter. If IPEM_MASS does not exist or is not set, the default as defined in the attribute map will be saved to Teamcenter.

PTC_COMMON_NAME

The integration handles this parameter uniquely for family table instances. This parameter appears in all family tables, but it is unlike any other parameter, as it can be unique for all instances – even nested instances. To read or write PTC_COMMON_NAME from a nested family table instance, the PTC APIs require that the instance be in session. If the instance is not itself in session as a top-level object and is not in session because it is used as a component of some other parent model, then the integration forces it into session during save.

To avoid the performance impact of loading models into session when PTC_COMMON_NAME is mapped, the integration does the following during open and save operations:

- When PTC_COMMON_NAME is mapped in the cadtopdm direction, instances are only loaded into session once the save has begun. Generation of the save dialog itself does not bring the instances into session, so the values shown in the save dialog for the parameter, or the target Teamcenter attribute, may not match what is in the CAD model and what will ultimately be saved to Teamcenter. For example, here is the 2nd level of a nested family table, with different values for each of the two instances:

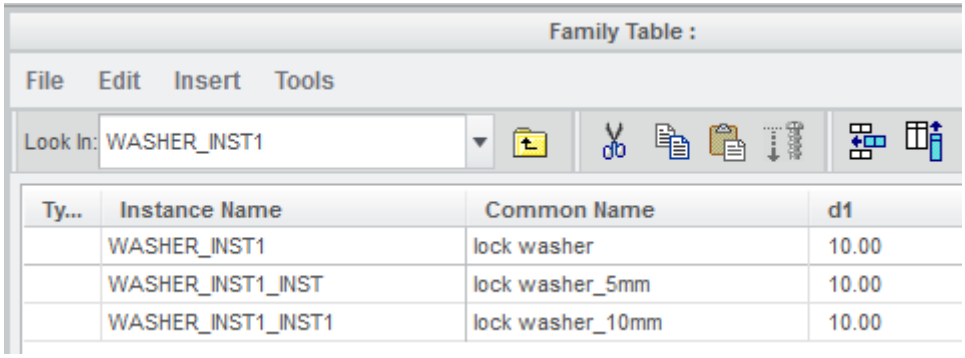


Figure 1 - Nested family table

In this case, the PTC_COMMON_NAME parameter is mapped to Item Name in Teamcenter. There is also a default value specified in the attribute map:

```
<attribute>
    <cad_name value="PTC_COMMON_NAME"/>
    <pdm_name value="item_name"/>
    <default type="String">NO_PTC_COMMON_NAME</default>
    <direction value="both"/>
</attribute>
```

When the generic part is saved to Teamcenter, none of its family table instances are in session. The user wants to save all instances to Teamcenter, so he presses the **Family Tables | Add** button, and the instances are added to the dialog. However, at this time, they are not brought into session, and so the value of PTC_COMMON_NAME cannot yet be read accurately. Instead, the integration displays the default value, from the attribute map definition, as the target Item Name attribute value:

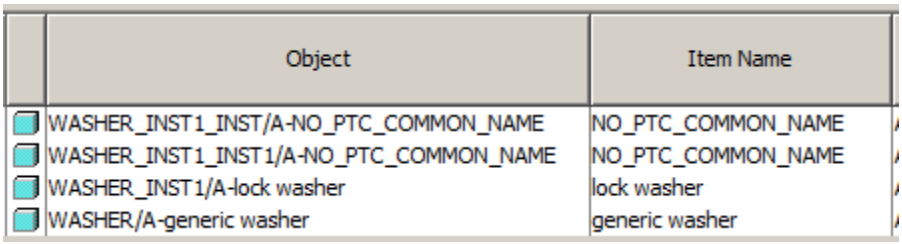


Figure 2 - Integration save dialog

If no default is configured in the attribute map, the actual value displayed will depend upon whether a default is available for the target property. In the case of Item Name, the displayed value will be the Item ID. This is because Item Name is derived from Item ID whenever it is not otherwise specified. Other properties may be displayed in the save dialog as blank values.

After the save to Teamcenter is finished, the correct parameter values will have been mapped to the Teamcenter Item Name property:

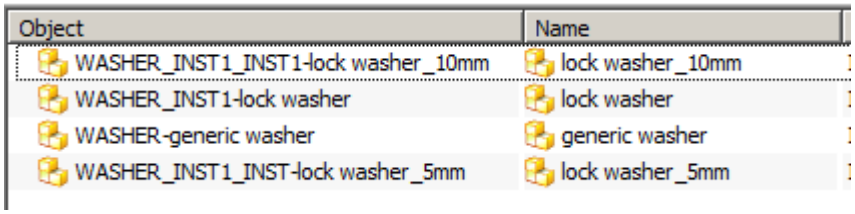


Figure 3 - Attribute values in Teamcenter

- When `PTC_COMMON_NAME` is mapped in the pdmtocad direction, the attribute mapping will only occur for instances that are brought into session during the open operation. Of course, instances that are brought into session later can be updated using the **Teamcenter | Update Attributes** function.

`PROI_REVISION` Creo® APIs does not allow writing to this parameter, so attribute mappings involving this parameter can only be configured in the cadtopdm direction. Other PROI parameters have similar behavior.

`MODEL_CHECK` Creo® can be configured to create and update this model parameter when ModelCHECK is run, and the parameter can then be mapped to a Teamcenter property, typically on the dataset or on the item revision. If the mapped Teamcenter property is included in an operation descriptor, it will appear on the corresponding dialog and may be editable during **Save** operations. Any edits made by the user will be ignored by the integration, and the value will always be copied from the Creo® model parameter. This parameter should never be mapped in the **pdmtocad**, or **both**, direction.

`MC_CONFIG` Creo® can be configured to create and update this model parameter when ModelCHECK is run, and the parameter can then be mapped to a Teamcenter property, typically on the dataset or on the item revision. If the mapped Teamcenter property is included in an operation descriptor, it will

	appear on the corresponding dialog and may be editable during **Save** operations. Any edits made by the user will be ignored by the integration, and the value will always be copied from the Creo® model parameter. This parameter should never be mapped in the **pdmtocad**, or **both**, direction. Creo® APIs do not allow writing to this parameter, so attribute mappings involving this parameter can only be configured in the cadtopdm direction. Other Creo® parameters have similar behavior.
`MC_ERRORS`	Creo® can be configured to create and update this model parameter when ModelCHECK is run, and the parameter can then be mapped to a Teamcenter property, typically on the dataset or on the item revision. If the mapped Teamcenter property is included in an operation descriptor, it will appear on the corresponding dialog and may be editable during **Save** operations. Any edits made by the user will be ignored by the integration, and the value will always be copied from the Creo® model parameter. This parameter should never be mapped in the **pdmtocad**, or **both**, direction.
`MC_MODE`	Creo® can be configured to create and update this model parameter when ModelCHECK is run, and the parameter can then be mapped to a Teamcenter property, typically on the dataset or on the item revision. If the mapped Teamcenter property is included in an operation descriptor, it will appear on the corresponding dialog and may be editable during **Save** operations. Any edits made by the user will be ignored by the integration, and the value will always be copied from the Creo® model parameter. This parameter should never be mapped in the **pdmtocad**, or **both**, direction.

pdm_name

`<pdm_name value="`*keyword*`"|"`*class*`[:`*type*`].`*name*`"/>`

The `pdm_name` tag specifies the property's name in Teamcenter. This name can be a keyword for certain special properties, or it can be a more general expression that gives the property's name and the class and type of object to which the property is attached.

Note

The Teamcenter property specified by the pdm_name tag may appear in the MultiFieldKey business object constant for the specified item type when the direction of the attribute mapping is either cadtopdm or both. If the values of all such mapped CAD parameters match the corresponding properties in Teamcenter, then the integration will consider the model to already exist in Teamcenter and will proceed accordingly. If one or more mappings to key Teamcenter properties results in a value mismatch, then the integration will consider the model new to Teamcenter and will process it as a first-time save.

When only a single keyword is given for the value of the `pdm_name`, it may be one of the following:

`dataset_desc`	The attribute is the dataset description.
`item_desc`	The attribute is the item description.
`item_name`	The attribute is the item name.
`item_id`	The attribute is the item ID.
`item_revision_id`	The attribute is the revision.
`item_revision_desc`	The attribute is the revision description.
`item_type`	The attribute is the item type. Since an item's type cannot change after the item has been created, this is ignored when copying attribute values to an existing item.

To specify attributes other than the special ones listed above, use the notation *class*`.`*name* for the `pdm_name` value, where *class* indicates where the attribute will be found, such as on an item, and *name* is the name of the attribute. The classes of Teamcenter objects currently supported are:

`Dataset`	The attribute is a dataset property.
`Item`	The attribute is an item property.
`Item Master`	The attribute is one of the form properties in the item master. Note that there is a single space in this class name.
`Item.Form`	The attribute is a property in a form attached to an item. The form must be based on a POM class derived from the Form class. If more than one form is attached to an item, the first one that matches the *class* and *type* (see below) is used. Forms that are stored as files are not supported.
`ItemRevision`	The attribute is an item revision property.
`ItemRevision Master`	The attribute is one of the form properties in the item revision master. Note that there is a single space in this class name.
`ItemRevision.Form`	The attribute is a property in a form attached to an item revision. The form must be based on a POM class derived from the Form class. If more than one form is attached to an item revision, the first one that matches the *class* and *type* (see below) is used. Forms that are stored as files are not supported.

For example, to specify the `project_id` attribute in the item revision master form, the name would be

```
<pdm_name value="ItemRevision Master.project_id"/>
```

An optional type can be specified using the notation *class*`:`*type*`.`*name*. A property `object_desc` that will be used only for items of type `Part` would be named

```
<pdm_name value="Item:Part.object_desc"/>
```

Types may also be included in the compound notation used to specify a general form property. For example, suppose items of type `Part` can have a form of type `PartData`. A property `data1` in a `PartData` form would be named

```
<pdm_name value="Item:Part.Form:PartData.data1"/>
```

When using the *class*.*name* or *class*:*type*.*name* notation, *name* must be the property's real name, not its display name. You can find the real name of a property in the BMIDE application.

Note

When a keyword exists for the attribute you want to map, use the keyword. Do not use the class.name or class:type.name notation when a suitable keyword is available, otherwise items might not be created or revised correctly.

The `pdm_name` tag is required, and it must be the second tag in an `attribute` definition, following the `cad_name` tag. It has no default.

Note

Errors or performance degradation could occur if the properties referenced as pdm_name are not included in the policy file. Specifically, "assertion" failures may appear in the txdlog file if a property is missing from the policy file.

Teamcenter runtime properties are not supported. Furthermore, Teamcenter reference properties, other than **uom_tag**, may not be used as either the source or destination for attribute mapping, in either direction.

default

`<default type="Double"|"Integer"|"String"|"Boolean">`*data*`</default>`

The `default` tag specifies a default value to assign to the destination property when the source parameter is missing. The value that is assigned to the destination property is *data*. Its type may be `"Double"`, `"Integer"`, or `"String"` (note that double quotes are required around the type as shown, but should not enclose *data*). The `"Boolean"` type may be used for Creo® Yes/No parameters and Teamcenter logical properties. When the type is `"Boolean"`, *data* must be either `true` or `false`.

The `default` tag is optional. If the source parameter is missing and no default is defined, the attribute is skipped. The `default` tag applies only to Teamcenter properties; therefore, it is only meaningful for attributes mapped in the `cadtopdm` direction.

missing_attribute_action

`<missing_attribute_action value="`**`create`**`"|"ignore"/>`

The `missing_attribute_action` tag specifies what to do when the destination parameter is missing. Its value may be one of the following strings (these strings must be enclosed in double quotes as shown).

`"create"`	The destination parameter is created if it does not exist. This is the default.
`"ignore"`	The attribute is skipped if the destination parameter does not exist.

The `missing_attribute_action` tag applies only to Creo® parameters; therefore, it is only meaningful for attributes mapped in the `pdmtocad`, or `both`, direction. It is ignored when the destination is a Teamcenter property.

direction

```
<direction value="both"|"cadtopdm"|"pdmtocad"/>
```

The `direction` tag indicates whether to copy the attribute only from Teamcenter to Creo®, only from Creo® to Teamcenter, or in both directions. Its value may be one of the following strings (these strings must be enclosed in double quotes as shown).

`"both"`	The attribute is copied from Teamcenter to Creo® when the model is opened, and it is copied from Creo® to Teamcenter when the model is saved. This is the default. If the attribute appears in the Properties Panel of a **Save** dialog, any modification made in the dialog will be written into the CAD file during save.
`"cadtopdm"`	The attribute is copied only from Creo® to Teamcenter when the model is saved.
`"pdmtocad"`	The attribute is copied from Teamcenter to Creo® when the model is opened from Teamcenter. The attribute will also be copied from Teamcenter to Creo® during **Save**, under the following conditions: • The Creo® property does not already exist, and the Teamcenter attribute has a non-null value. In this case, the Creo® property will be created and populated with the value from Teamcenter, during **Save** or **Revise**, if `missing_attribute_action = "create"`[1]. • The Creo® property does already exist. The Teamcenter attribute value, whether null or non-null, will be written to Creo® during **Save** or **Revise** . If the attribute appears in the Properties Panel of a **Save** dialog, any modification made in the dialog will be written into the CAD file during save.

Note

Properties that are mapped in the pdmtocad direction are synchronized during **Save** for performance reasons. Because the intent of the mapping is to drive Creo® property values from Teamcenter, the integration performs the mapping during **Save** so that it is not required on a subsequent **Open** from Teamcenter. Synchronizing Creo® properties during **Open** is relatively more expensive than it is during a **Save** operation.

Note that two `attribute` definitions can be provided when different behavior is needed for each direction in which the attribute is copied. One definition may specify the `cadtopdm` direction and the other definition may specify the `pdmtocad` direction.

[1] Note that the default value of `missing_attribute_action`, if not specified in the attribute_map, is "`create`".

constraint

```
<constraint value="any"|"master"|"non_master"|"reserved"|"use_locale"/>
```

The `constraint` tag restricts attribute copying to models with certain characteristics. Its value may be one of the following strings (these strings must be enclosed in double quotes as shown).

`"any"`	There is no restriction on copying the attribute. This is the default.
`"master"`	The attribute should be copied only if the model "owns" its file, which in the case of Creo® refers to models that are not family table instances. Use this constraint value to prevent the attribute from being copied to or from family table instances.
`"non_master"`	The attribute should be copied only if the model does not "own" its file. Use this constraint value to copy the attribute when the model is a family table instance, and to prevent the attribute from being copied for all other models.
`"reserved"`	The attribute should be copied only if the user reserves the model. Use this constraint value to avoid modifying models that cannot be checked in. This constraint applies only to the `pdmtocad` direction. It is ignored when attributes are copied from Creo® to Teamcenter.
`"use_locale"`	The locale specific Teamcenter value is mapped to the Creo® property, rather than the master value. The `use_locale` constraint is only valid for pdmtocad mappings. If the Teamcenter property is a Multifield key property, then the property will appear in the dialog with its master value, not its locale value. The `use_locale` constraint will degrade the performance of **Open** and other integration operations because it will always force synchronization of property values.

The `constraint` tag may be used more than once. For example, to make sure an attribute is copied only to models that are checked out for modification, and not copied to family table instances, use two constraint tags:

```
<constraint value="reserved"/>
<constraint value="master"/>
```

truncate

```
<truncate value="true"|"false"/>
```

The `truncate` tag specifies whether an attribute should be truncated when mapping from CAD to PDM. Should a Creo® parameter value be longer than the corresponding Teamcenter attribute definition, the attribute can be truncated to the length of the Teamcenter property.

`"true"`	Truncate the attribute prior to writing to Teamcenter.
`"false"`	Default value. The attribute is not truncated and mapping in the cadtopdm direction is silently skipped.

When the `truncate` tag is used, it is necessary to fully qualify the `pdm_name`. For example if a site has a custom item type called “PRO_Geometry” with corresponding item and item revision master forms named “PRO_GeometryMaster” and “PRO_GeometryRevisionMaster” then the following `pdm_name` tag should be used:

```
<pdm_name value="ItemRevision.Form:PRO_GeometryRevisionMaster.drawing_number"/>
```

The following abbreviated syntax may not be used with the truncate tag:

```
<pdm_name value="ItemRevision Master.drawing_number"/>
```

Attribute Mapping and Family Tables

A parameter used in a Creo® family table can be mapped to Teamcenter properties like any other parameter. However, you should keep the following points in mind when working with family tables.

The values assigned to family table instance parameters can be controlled only at a single level of the family table. This is a limitation of Creo® itself. For example, suppose that a generic part has a string parameter called `DESCRIPTION`, with a value of “generic washer”:

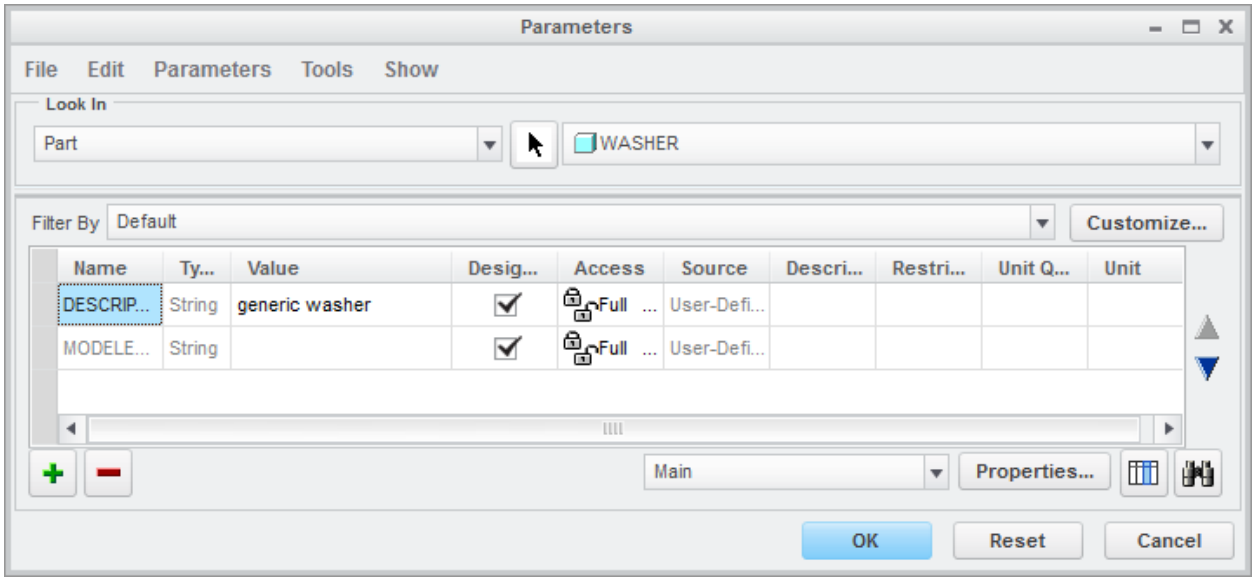


Figure 4 - Creo® parameter dialog

When you add the first level of family table instances to the generic part, you can include the `DESCRIPTION` parameter as a column, and assign different parameter values to each instance:

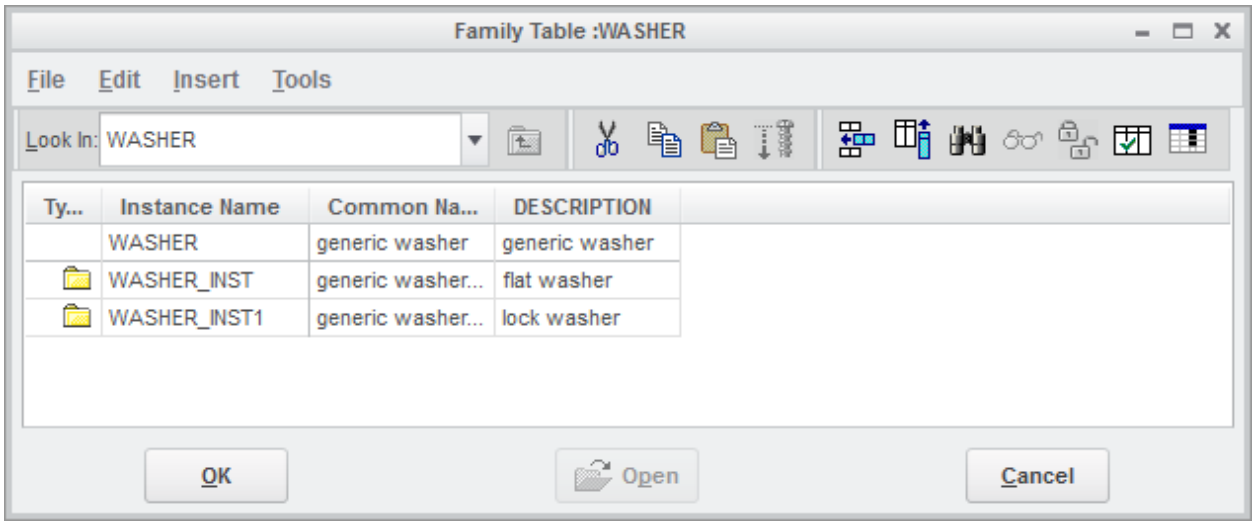


Figure 5 - Family table definition dialog - level 1

And the DESCRIPTION parameter can be mapped to a Teamcenter attribute, such as the Item Description:

```
    <attribute>
        <cad_name value="DESCRIPTION"></cad_name>
        <pdm_name value="item_desc"></pdm_name>
        <missing_attribute_action value="create"></missing_attribute_action>
        <direction value="both"></direction>
</attribute>
```

When you save this part and its family table to Teamcenter, the attribute mapping works as expected: the generic and each instance are given a unique Item Description:

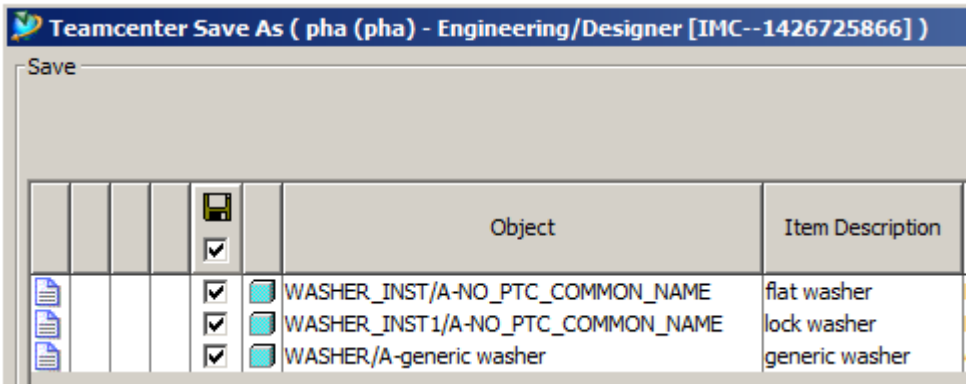


Figure 6 - Teamcenter save dialog - level 1

If a second level is added to the family table, the user can no longer assign the value of the DESCRIPTION parameter. Instead, the value of each 2nd-level instance (and of instances at all lower levels) is inherited from its parent generic model. Here we are attempting to add `DESCRIPTION` as a column to the 2nd level of the family table, but Creo® does not allow it:

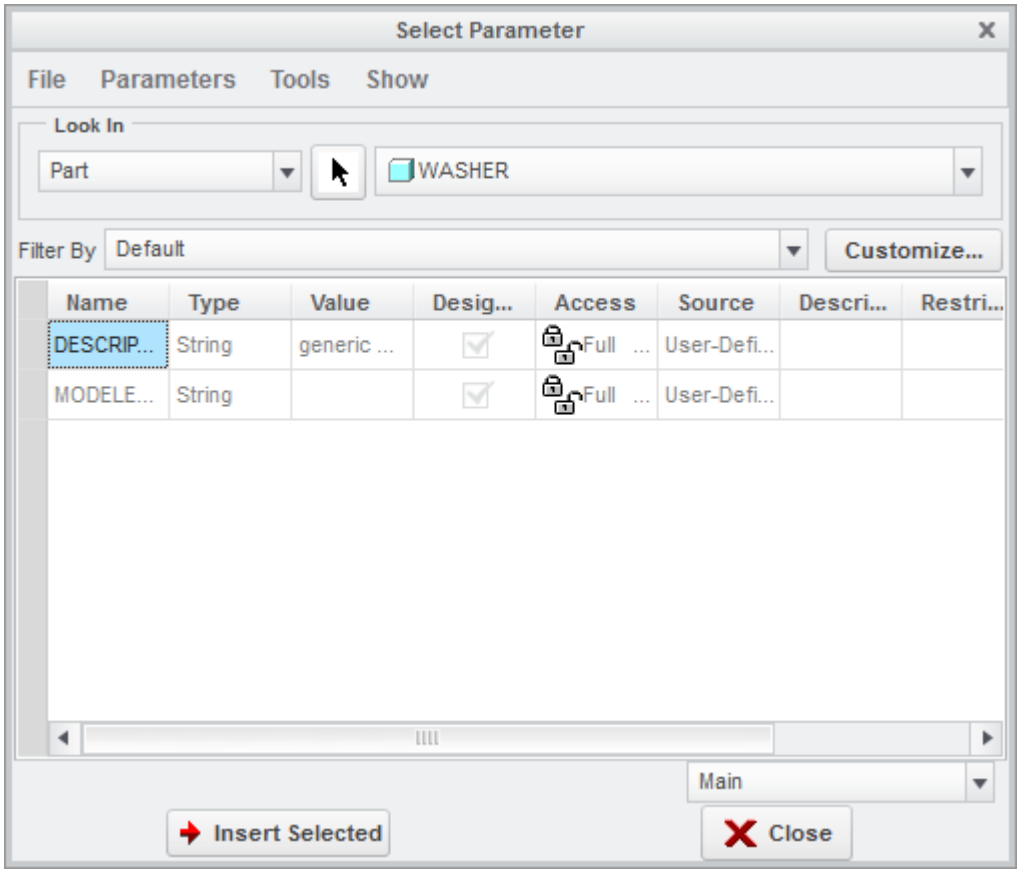


Figure 7 - Insert parameter as family table column

Figure 8 - Creo® warning message

Consequently, when the 2nd-level instances are saved to Teamcenter, their Item Description properties will be driven by the `DESCRIPTION` parameter of their generic model, resulting in non-unique Teamcenter property values:

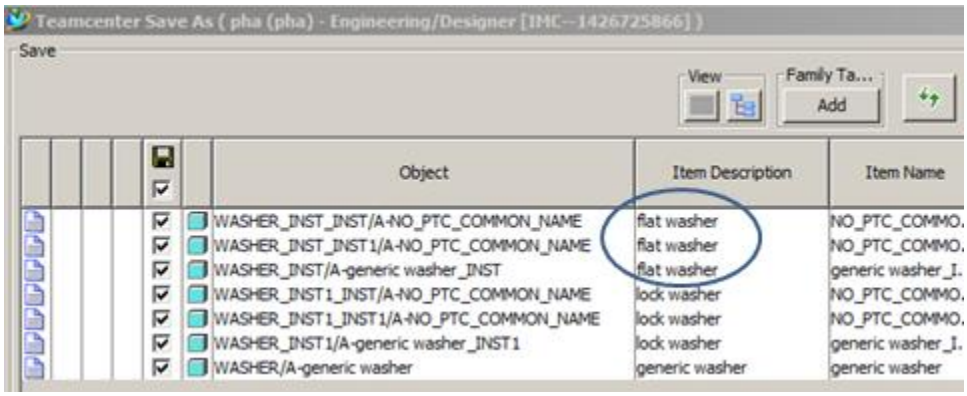


Figure 9 - Teamcenter save dialog - level 2

Although the Teamcenter result is consistent with the CAD models' parameter values, the usefulness of cadtopdm attribute mapping is limited when nested family tables are involved.

Note

The special parameter PTC_COMMON_NAME is an exception to the Creo® rule against unique parameter values for nested family table instances. Here is the second level of the family table, with different values for PTC_COMMON_NAME assigned to the 2nd level instances:

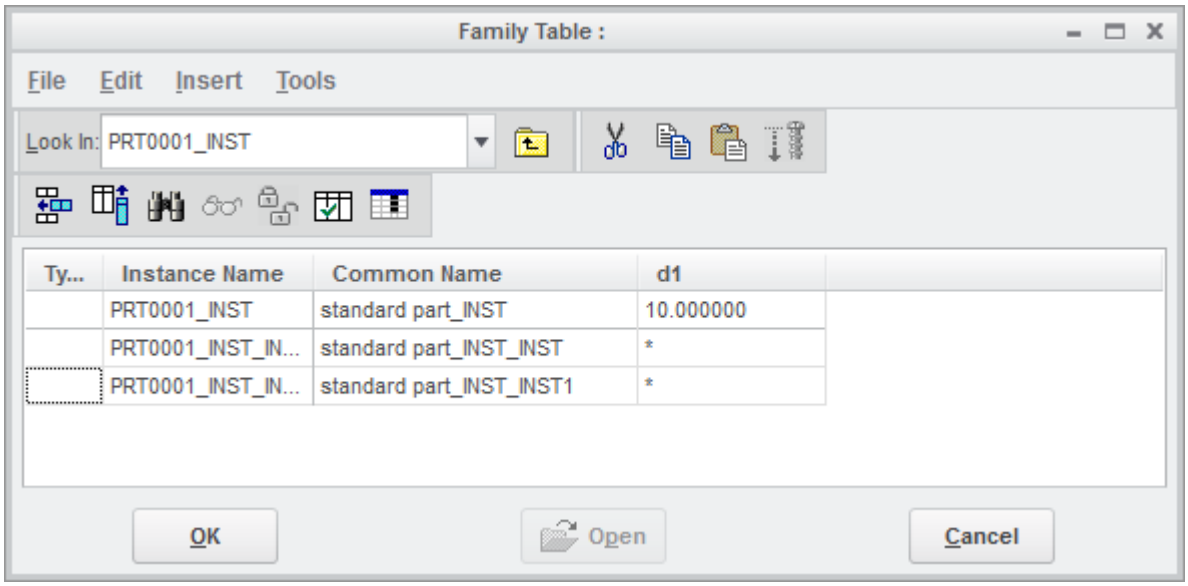


Figure 10 - Family table definition dialog - 2nd level

Note

PTC_COMMON_NAME can be mapped to Teamcenter, like any other CAD parameter. However, for Teamcenter to read its value correctly, the instance must be brought into session. This may have a significant performance impact on save, for large, nested family tables, or where instance regeneration time is long due to size or complexity of the CAD models.

Attribute mapping from Teamcenter to Creo® (pdmtocad direction)

- When a parameter is a family table column, a separate instance of the parameter is created for each member of the family table, but only at the first level of the family table. Changing the parameter's value for one 1st level family table instance does not affect the value of that parameter for other 1st level family table instances. Instances at the 2nd and lower levels of the family table are driven by the parameter values of their parent generics, as in the preceding example.
- Parameters that are not family table columns are shared by all members of the family table. Changing the parameter's value affects every member.

Let's consider what happens when data from the previous section is saved to Teamcenter, then opened again into Creo®. The attribute map remains the same:

```
<attribute>
    <cad_name value="DESCRIPTION"></cad_name>
    <pdm_name value="item_desc"></pdm_name>
    <missing_attribute_action value="create"></missing_attribute_action>
    <direction value="both"></direction>
</attribute>
```

After the initial save to Teamcenter, the Item Description values were updated in the Rich Client, to make them unique. Therefore, in the **Teamcenter | Open** dialog we see different Item Description values for the top-level generic and each instance:

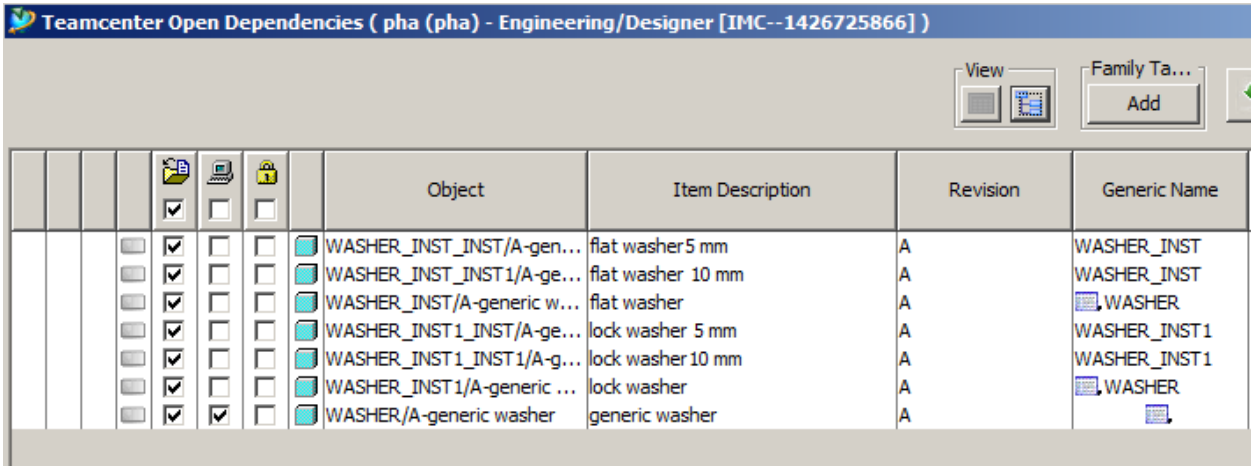


Figure 11 - Open from Teamcenter, showing unique Item Description values

When the CAD models are opened in the Creo® session, the Creo® constraint will take effect and the value of each lower-level instance will be forced to the same value as its parent generic:

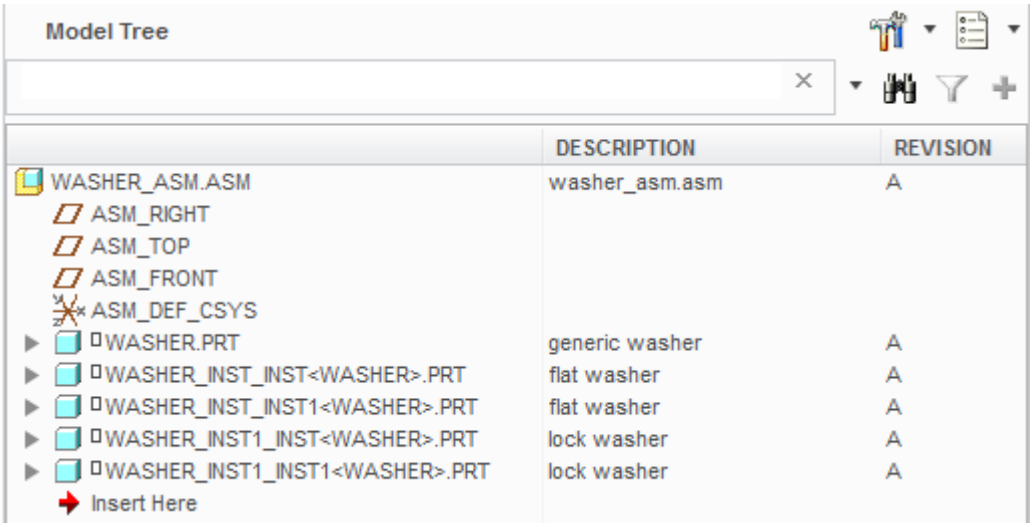


Figure 12 - Creo® model tree with parameter displayed

The actual value assigned to the parameters will depend upon which object's attribute is mapped last.

What happens when the CAD model's family table does not include the CAD parameter at all? Note that the `missing_attribute_action` is `"create"`, which means the `DESCRIPTION` parameter will be created in the Creo® model if it does not already exist when the Item Description is copied from Teamcenter. If the model is a family table instance, this action will create a parameter named `DESCRIPTION` that is shared by all members of the family table. This could lead to unexpected results since the generic and every instance will show the same value for the `DESCRIPTION` parameter.

Customers may want to consider setting the `constraint` tag to `"master"` or setting `missing_attribute_action` to `"ignore"` to avoid creating a potentially confusing shared parameter.

Loading the Attribute and Column Mappings into Teamcenter

The attribute mappings defined by the **attribute_map** and **column_map** tags in the **ipem.xml** file must also be loaded into the Teamcenter database. Load the mappings with the following procedure. When performing these steps, use the same **ipem.xml** file that will be used in the runtime environment.

Note

The integration leverages Teamcenter attribute mapping support for improved performance. The complete syntax and requirements for this Teamcenter functionality are described in the Teamcenter documentation. The integration provides a utility and simplified documentation, here, to automate the configuration process as much as possible. This may not be sufficient for all customers and all configurations, particularly those involving custom Item types and custom relations. In such cases, please refer to the Teamcenter documentation on attribute mapping.

The attribute mappings defined by the **attribute_map** tag, and the column mappings defined by the **column_map** tag in the **ipem.xml** file must always match the corresponding maps in the Teamcenter database. A four-step process is used to create and merge the mappings into Teamcenter.

It is only necessary to perform this step once per site, during initial installation. If attribute mappings or column mappings are changed, this step must be repeated, again once per site.

1. Convert the attribute_map and column_map tags defined in ipem.xml into a text file.
    a. Open a Teamcenter command shell and change your working directory to the %TC_ROOT%\install\ipem location. This directory was created by the integration's server installation process.
    b. Run **create_attr_mappings.bat** (**create_attr_mappings.sh** for Bourne shell or **create_attr_mappings** on UNIX; you may need to change the permissions on the file) to create a text file from the mappings definitions in ipem.xml:

```
create_attr_mappings –u=infodba -p=infodba -i=ipem.xml –
o=ipem_attr_mappings.txt
```

    If item types other than **Item** are used with the integration, then those item types must be specified using the **–itemtypes** argument. -**itemtypes** is a colon-separated list. If the item type name contains spaces, it must be enclosed by double quotes:

```
create_attr_mappings –u=infodba -p=infodba -i=ipem.xml –
o=ipem_attr_mappings.txt –itemtypes=Part:Document:Functionality:"Custom
Requirement":"Requirement"
```

2. Extract all existing mappings from Teamcenter and make a backup copy.
    a. From the Teamcenter command shell, use the **export_attr_mappings** utility to create an **existing_attr_mappings.txt** file:

```
export_attr_mappings –u=infodba -p=infodba -g=dba  -
file=existing_attr_mappings.txt
```

Note

Warning messages, such as the example below, may be ignored:

```
WARNING: Item Master form - no master qualifier found, adding master=iman
for attribute =  : Item.GRM(IMAN_master_form,Item Master)…
```

    b. Make a backup of the exported file.

Note

The Teamcenter **import_attribute_mappings** script first deletes all existing mappings from Teamcenter and then imports the new mappings from the provided input file. The merge process described below is required to prevent the loss of existing non-integration attribute mappings.

1. Merge the integration mappings with the existing mappings. If **existing_attr_mappings.txt** contains integration mappings, then remove the integration mappings from **existing_attr_mappings.txt** and then concatenate **ipem_attr_mappings.txt** and **existing_attr_mappings.txt** to create **merge_attr_mappings.txt.**
2. Import the mappings into Teamcenter
    a. From the Teamcenter command shell, run the **import_attr_mappings** utility to import the mappings into Teamcenter:

```
import_attr_mappings –u=infodba -p=infodba -g=dba -
file=merge_attr_mappings.txt
```

b. If import fails, review the failures, correct, and import again.

Properties display in the Save and Teamcenter New Dialogs

The **Save and Teamcenter New** dialogs have two properties panels: **Required Properties** and **Optional Properties**. Initially, the **Required Properties** panel contains only the system properties required by OOTB Teamcenter. These system properties are **Item Type**, **Item ID**, **Item Revision**, **Item Name**, and **Model Name**. The **Optional Properties** panel tabs contain properties that have Teamcenter *operation descriptors*. By default, only a few properties have *operation descriptors*.

It is possible to change the contents of the properties panels by attaching Teamcenter *operation descriptors* to objects, using the BMIDE application. When an *operation descriptor* has been designated as `visible`, the corresponding property will appear in one of the **Optional Properties** tabs. Setting an *operation descriptor* to `required`, as well as `visible`, in BMIDE will cause the property to appear in the **Required Properties** panel. All required properties must have a value before the model can be saved for the first time. The **not modifiable** icon ⊘ appears in the left most column of the **Save** dialog when the user has selected that model for save but has not provided values for all required properties.

Required and **Optional** properties can be included in attribute mappings. When an attribute mapping exists for a property, its value in the properties panel will initially be determined by the value of the mapped Creo® parameter. Mapped parameters have the advantage of being visible to the user from within Creo®. When a user assigns a value to a mapped property in the properties panel, that value is written back into the Creo® file when the **Save** button is selected. Read more about attribute mapping in the ipem.xml configuration section of this document.

Note

Operation descriptors can differ between Item Types.

Models existing in Teamcenter without values for required properties can be saved without specifying a value.

Do not use a function in an attribute mapping to a property that has a create descriptor. The functional value will persist and user changes in the save dialog will not persist.

Limit required create descriptors to properties that CAD designers know at time of first save. An excessive number of required create descriptors might discourage designers from saving data to Teamcenter.

Configuring the Properties Panel for Teamcenter New and Save Operations

The properties panel in the **Save** and **Teamcenter New** dialogs is used to display and edit Teamcenter properties of Creo® models as they're being saved and created. The default layout and ordering of the panel may not be appropriate for all customers, and so a `properties_panel_style` map in the ipem.xml file may be used to override both the content and the sequencing of properties.

The integration first reads the master properties definition from Teamcenter, then uses the `properties_panel_style` map, if it exists, to alter the visibility and positioning of designated properties on the **Save** and the **Teamcenter New** dialogs.

Note:

The tags that control property visibility and ordering within the dialogs cannot be limited by item type. They will be applied to all OOTB and custom item types. When deciding which properties to include, be sure you have accounted for all item types that can be created by the Creo® integration. The list of available item types is controlled by the dataset_map in ipem.xml, using the allowed_item_types tag.

Required properties

Required properties appear in a flat list, undifferentiated by object type attachment. To specify the order of required properties, use the `order` attribute within the `required_properties` tag. System-level properties cannot be configured; these are:

- Item Type
- Item ID
- Item Revision ID
- Item Name

Any other custom or OOTB required property that appears in the **Required** section of the **Save** and **Teamcenter New** dialogs can be configured. For example, here is the **Save** dialog for a custom item type. The `properties_panel_style` map has not yet been added to the ipem.xml file.

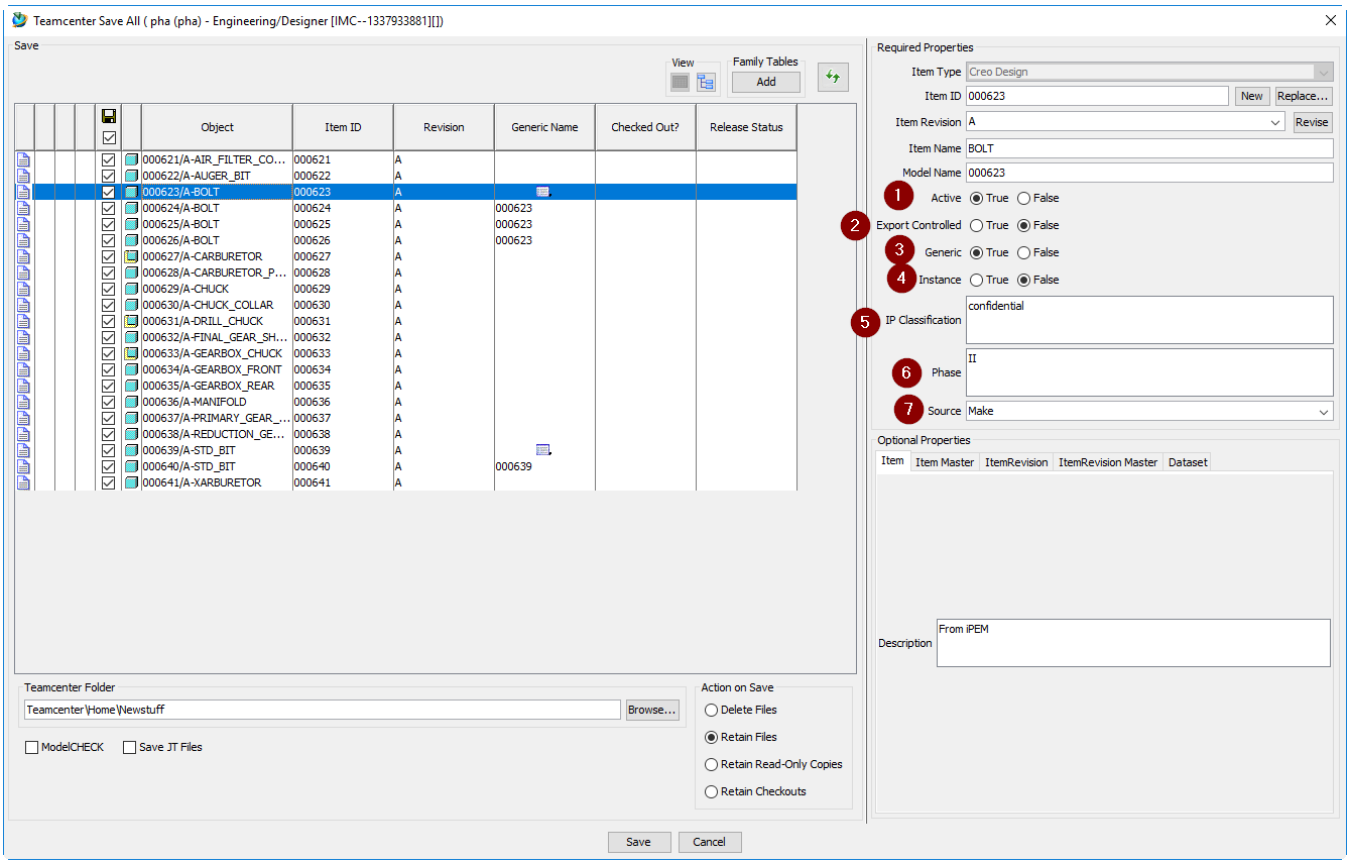


Figure 13 - Unconfigured Save dialog

The order in which OOTB and custom required properties appear (except for the four system-level properties), is alphabetical, and cannot be controlled by configuration in Teamcenter. This customer can reorder the non-system properties by adding the `properties_panel_style` map as follows:

```
<properties_panel_style>
      <required_properties
      order="Item.pa4Source:Item.pa4IPClassification:Item.pa4ExportControlled:
      ItemRevision.pa4Phase:ItemRevision.pa4Active:Item.pa4Generic:
      Item.pa4Instance"/>
</properties_panel_style>
```

Note:

Although the `properties_panel_style` map's syntax is derived from that of the `attribute_map`, use of the type qualifier is not supported. In the above example, `Item.pa4Source` is recognized as a valid property; `Item:MyCustomType.pa4Source` is not.

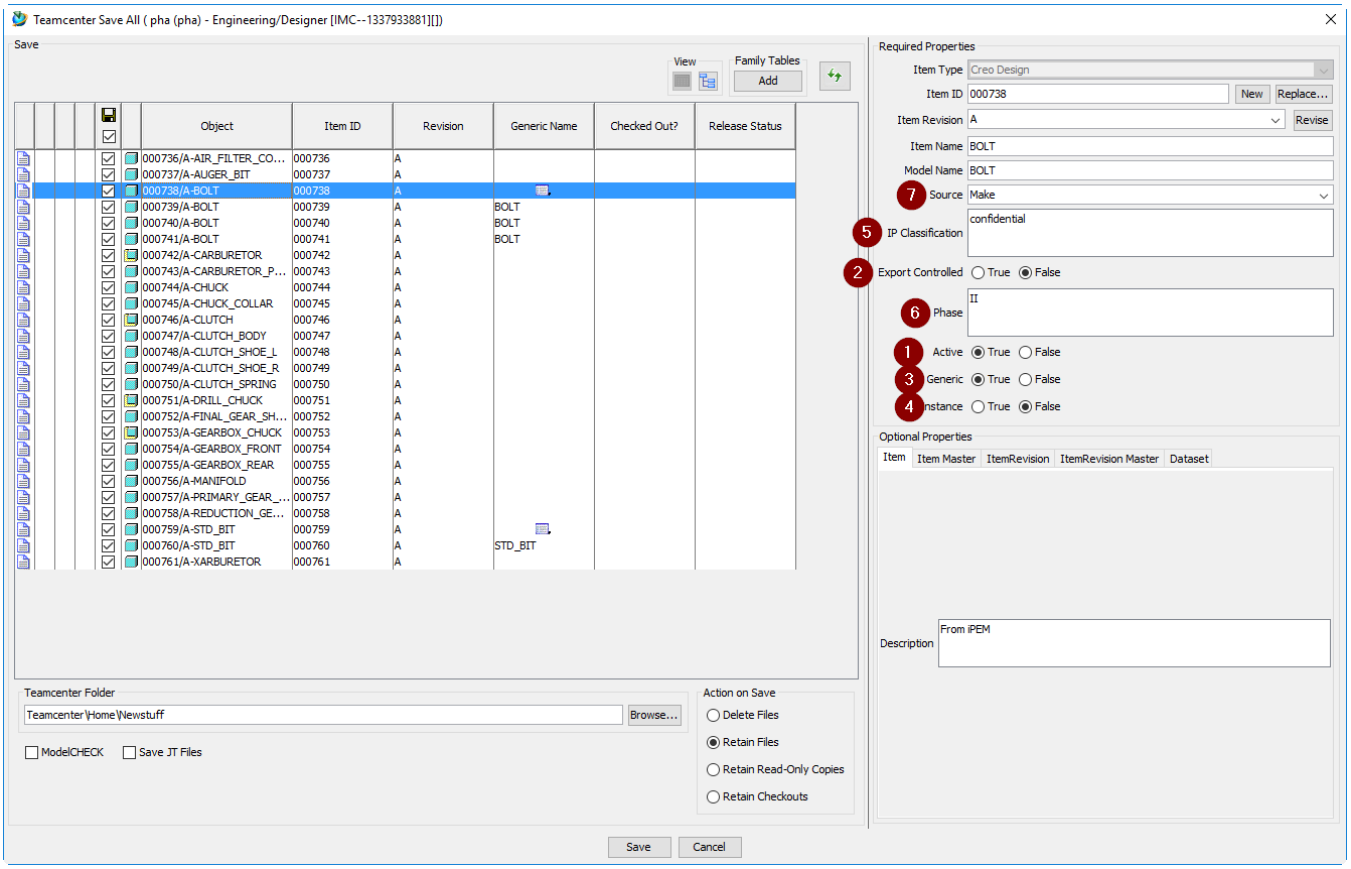


Figure 14 - Configured Save dialog

Note:

If a **Required** property is omitted from the `required_properties` list, it will not appear on the dialog, even though it is designated as **Required** by the Teamcenter operation descriptor. This may prevent

users from completing the **Save** or **Teamcenter New** operations from within Creo® unless other arrangements are made to automatically assign a value to all required properties.

Optional properties

Unlike **Required** properties, **Optional** properties are organized by object type attachment, using a tabbed layout. The visibility and ordering of the tabs, as well as the properties displayed on each tab, can be controlled by the `properties_panel_style` map.

To modify the ordering of optional property tabs, use the `order` attribute within the `optional_tabs` tag. The tag is a colon separated list of properties of the tabs. The valid tabs are: `Item`, `Item Master`, `ItemRevision`, `ItemRevision Master`, and `Dataset`. The order of the list defines the order in which tabs are displayed. If a tab is not included in the list, it will not appear on the dialog. For example, here is the **Optional** properties panel for a custom item type. The `optional_tabs` tag has not yet been added to the `properties_panel_style` map, and so the tabs appear in their default ordering.

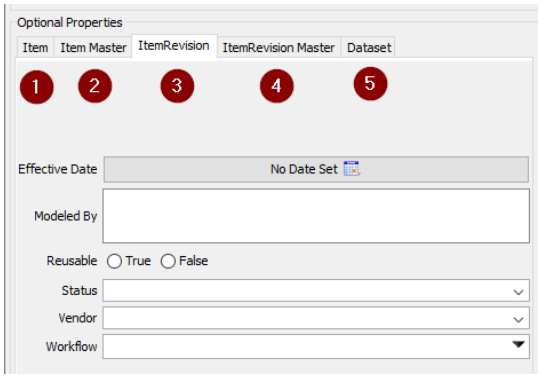


Figure 15 - Unconfigured Optional properties tabs

In this example, the custom item revision type has several custom properties that are important, and so the customer wants the Item Revision tab to be the first displayed. He also wants to suppress the display of the master form tabs. The optional_tabs tag shown below will accomplish this:

```
<properties_panel_style>
      <required_properties
      order="Item.pa4Source:Item.pa4IPClassification:Item.pa4ExportControlled:
      ItemRevision.pa4Phase:ItemRevision.pa4Active:Item.pa4Generic:
      Item.pa4Instance"/>
      <optional_tabs order="ItemRevision:Item:Dataset"/>
</properties_panel_style>
```

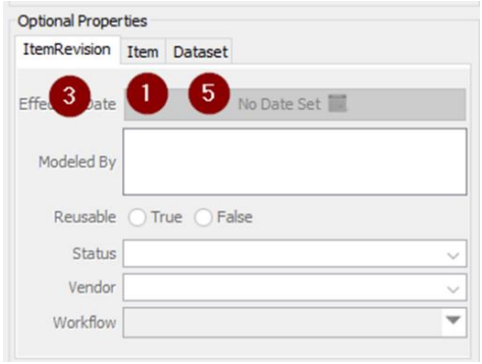


Figure 16 - Configured Optional properties tabs

Finally, the ordering of OOTB and custom **properties** on each tab can be controlled by tab-specific tags within the `properties_panel_style` map. As with **Required** properties, **Optional** properties appear by default in alphabetical order. Here is the Item Revision tab without any configuration:

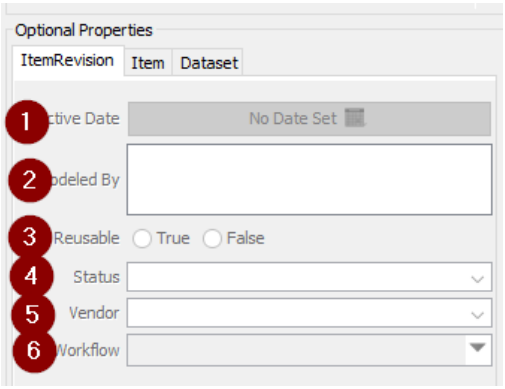


Figure 17 - Unconfigured Item Revision tab

The ordering of properties within each tab can be controlled by one of the following tags:

- `item_properties`
- `item_revision_properties`
- `item_master_properties`
- `item_revision_master_properties`
- `dataset_properties`

For example, the item revision tab's properties can be reordered as follows:

```
<properties_panel_style>
      <required_properties
      order="Item.pa4Source:Item.pa4IPClassification:Item.pa4ExportControlled:
      ItemRevision.pa4Phase:ItemRevision.pa4Active:Item.pa4Generic:
      Item.pa4Instance"/>
      <optional_tabs order="ItemRevision:Item:Dataset"/>
      <item_revision_properties
      order="pa4Vendor:pa4Reusable:pa4Status:pa4EffectiveDate:pa4ModeledBy"/>
</properties_panel_style>
```

Note:

It is not necessary to qualify each property name with its owning object type, because the naming convention applied to the tab-specific tags already provides the necessary information

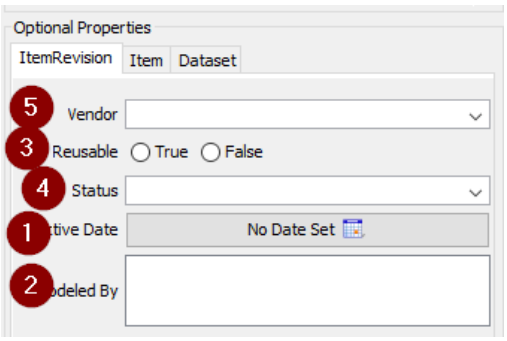


Figure 18 - Configured Item Revision tab

If a property is omitted from the tab-specific tag, it will not appear, regardless of its **Visible** designation in the Teamcenter operation descriptor definition. This is demonstrated in the example above, as the **Workflow** property is not displayed.

Note:

Although the above examples are based on the **Save** dialog, the **Teamcenter New** dialog is configured in the same way, by the `properties_panel_style` map.

Property naming rules display in the Save and Teamcenter New Dialogs

In the BMIDE it is possible to attach naming rules to properties. In the **Save** and **Teamcenter New** dialogs, when a user selects an Item Type with a naming rule attached to the Item ID, a drop-down list appears between the **Item ID** field and the **New** button so that the user can select the naming rule.

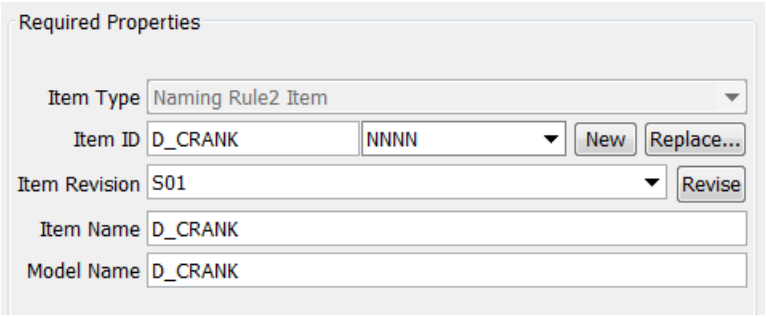


Figure 19 – Naming Rule NNNN has been selected

The model cannot be saved until the Item ID conforms to the selected naming rule. Users must select the naming rule before selecting the **New** or **Replace** buttons.

A naming rule may also be attached to the item revision ID, and will be displayed in the same way:

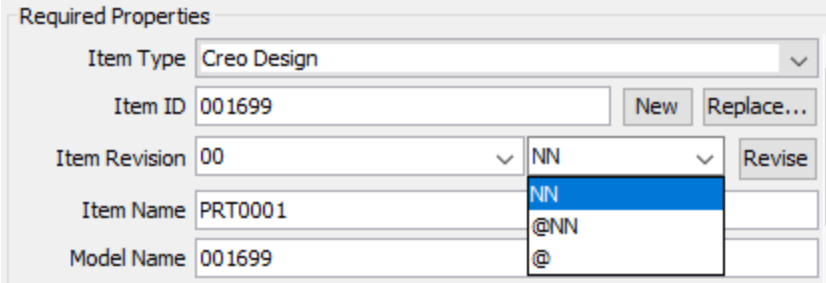


Figure 20 - Item revision ID naming rule

Configuring the ID Generator Dialog

Using BMIDE, an **ID Generator** can be attached to an Item Type to enable assignment of **Intelligent Part Numbers** (**IPN**). For Item Types that have an associated **ID Generator**, the integration displays the **ID Generator** dialog when the user clicks on the **New** button during **Save** or **Teamcenter→New** operations.

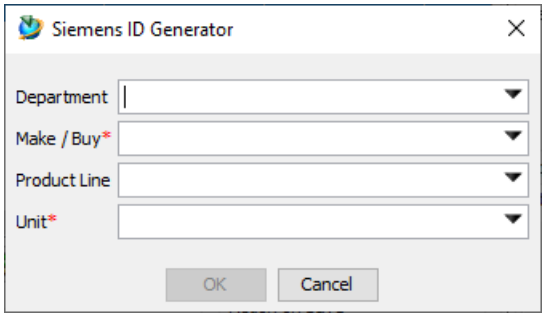


Figure 21 – Sample ID Generator Dialog

In the example above, “Siemens ID Generator” is the display name of the ID Generator. **Department**, **Make/Buy**, **Product Line**, and **Unit** are all properties of the ID Generator. In this definition, each of the properties has an attached **List of Values** (**LOV**). Other types can appear as well. The red asterisk next to the **Make/Buy** and **Unit** properties indicates that those fields are required.

Without additional configuration, the integration will display the ID Generator properties as they are reported by Teamcenter, in alphabetical order. If a different ordering is required, then add an **idgenerator_dialog_style** map to the ipem.xml file. The **ipn** tag specifies the name of the ID Generator, and the **order** tag controls the sequencing of the property fields within the dialog. For example, the property fields of the ID Generator shown in **Figure 21** could be rearranged using the following **idgenerator_dialog_style** map:

```
<idgenerator_dialog_style>
     <ipn name="ITI9_ipnIdGenSiemens"
     order="iti9_idGenDepartment:iti9_idGenProductLine:iti9_idGenMakeBuy:iti9_idGen
     Unit"/>
</idgenerator_dialog_style>
```

resulting in this modified dialog:

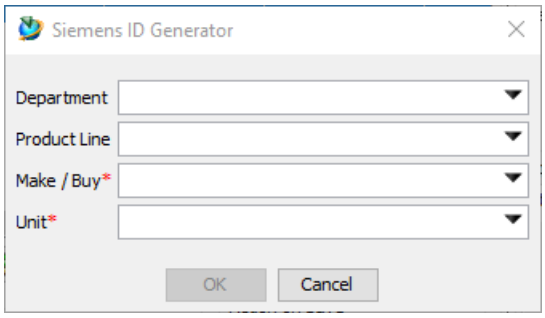


Figure 22 – Sample ID Generator Dialog with an `idgenerator_dialog_style` map

idgenerator_dialog_style map

```
<idgenerator_dialog_style>
```

The `idgenerator_dialog_style` map contains one or more `ipn` tags referencing ID Generator objects attached to item types that are used for managing Creo® models.

ipn tag

```
<ipn name="keyword" order="field 1 name[:field n name]"/>
```

The `ipn` tag specifies a single **ID Generator** object (a subclass of **Fnd0BaseIdGenerator** in the Teamcenter BMIDE schema) whose fields are to be reordered in the **ID Generator** dialog. The value of the `name` attribute is the internal BMIDE object name (not the display name) of the **ID Generator** business object. The `order` attribute is a colon separated list of property names belonging to the specified ID Generator object, in the order they are to appear on the dialog.

The height and width of the integration's **ID Generator** dialog can be specified by the `checkin.idgenerator.height` and `checkin.idgenerator.width` preferences in **ipem.properties**. Use these preferences to configure a dialog size that makes all **ID Generator** fields fully visible in the dialog.

Properties enabled in Save Dialog for Second Save[2]

Each row displayed in the **Save** dialog has an associated action context (Create New, Save, Revise or Save As). The properties displayed and enabled for each action context can differ. Teamcenter provides descriptors for **Create**, **Save As** and **Revise**, but does not provide a descriptor for **Save**. The integration honors the **Create**, **Save As** and **Revise** descriptors. For **Save**, the integration uses the **Create** descriptor to control which properties are displayed. The `save_enabled_properties` map in the ipem.xml file controls which properties are editable in the context of a **Save** operation. If the `save_enabled_properties` map does not appear in the ipem.xml file, then no properties may be changed in the **Save** dialog. In the following example, several properties of both the OOTB Item type and a custom type are made editable.

```
<save_enabled_properties>
     <item_type name="Item">
         <properties>
```

[2] "Second Save" is defined as saving a new dataset version of the current item revision of a model.

```
            <pdm_property name="Item.object_desc"/>
            <pdm_property name="Dataset.object_desc"/>
        </properties>
    </item_type>
    <item_type name="ITI9_PropItem">
        <properties>
            <pdm_property name="Item.object_desc"/>
            <pdm_property name="Dataset.object_desc"/>
            <pdm_property name="Item:ITI9_PropItem.iti9_name"/>
            <pdm_property
            name="ItemRevision:ITI9_PropItemRevision.Form:ITI9_PropItemRevisionMast
            er.user_data_1"/>
    </properties>
    </item_type>
</save_enabled_properties>
```

save_enabled_properties tag

`<save_enabled_properties>`

The `save_enabled_properties` tag contains one `item_type` tag per item type for which properties will be editable in the **Save** dialog.

item_type tag

`<item_type>`

The `item_type` tag specifies a single item type whose properties will be editable in the **Save** dialog.

properties tag

`<properties>`

The `properties` tag contains the properties to be enabled during save action.

pdm_property tag

`<pdm_property name="`*keyword*`"|"`*class[:type].name*`"/>`

The `pdm_property` tag specifies a Teamcenter property to be enabled during **Save**. The name is a general expression that gives the property's name and the class and type of object to which the property is attached. The general expression syntax is the same as documented for the `pdm_name` tag in the attribute mapping section of this document.

User Defined Columns

User defined columns are used to display attributes in the Integration's dialogs which are not available as keyword column tags in the `ipem.properties` file preferences. Such columns may be associated with the Teamcenter Item, Item Revision, Dataset or Form classes, or to properties of the Creo® models. They can be added to any table.columns.* preference in ipem.properties. For example, the following settings cause two user defined columns to be displayed:

```
table.columns.std = itemrev:revdesc:documentname:reservedby:releasestatus
```

```
table.columns.checkin =
itemid:itemrev:creomaterial:documentname:reservedby:releasestatus
```

The corresponding entries in the `ipem.xml` file's column_map are these:

```
<column_map>
      <column tag="creomaterial">
            <display_name value="Material"/>
            <cad_name value="Material"/>
      </column>
      <column tag="revdesc">
            <display_name value="Item Revision Description"/>
            <pdm_name value="ItemRevision.object_desc"/>
      </column>
</column_map>
```

A Teamcenter OOTB or custom properties may be displayed in a column in any integration dialog. In the example above, the Teamcenter Item Revision Description is displayed in a column titled **Item Revision Description**, and it is included in all dialogs where not overridden by a dialog-specific preference, because its tag, **revdesc**, is included in the `table.columns.std` preference. Here, the property is displayed in the **Open** dialog when an item revision is visible:

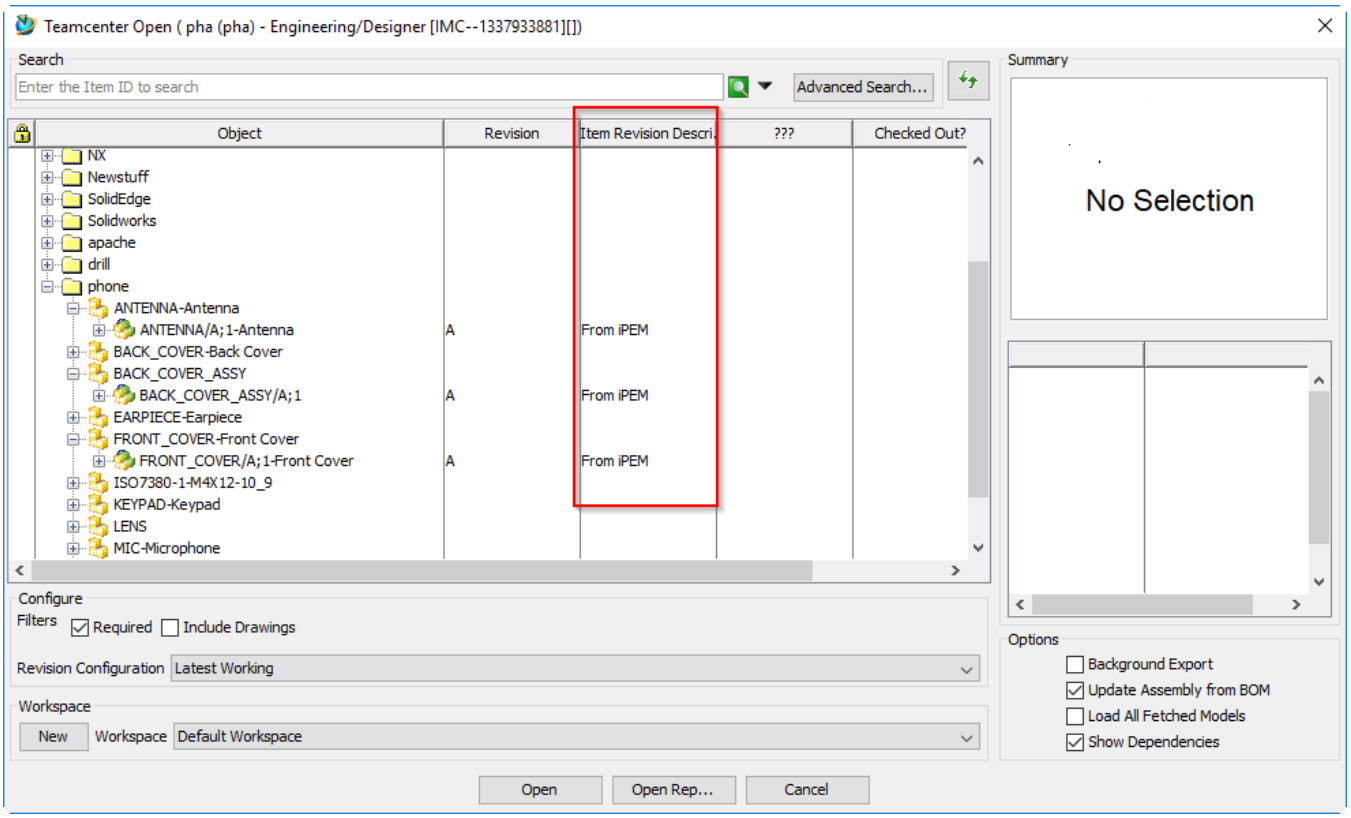


Figure 23 - Teamcenter property displayed in a column on the Open dialog

A column map that displays a Teamcenter property must be loaded into Teamcenter by following the instructions in Loading the Attribute and Column Mappings into Teamcenter. This enables the integration to leverage the Teamcenter attribute mapping framework to display column values. It does

not mean that the Teamcenter property displayed in a column must also be mapped to a SOLIDWORKS® model property (although it may be mapped, if desired).

Note:

If a user defined column that displays the value of a Teamcenter property is added or changed, you must repeat the attribute mapping load process described in Loading the Attribute and Column Mappings into Teamcenter.

A Creo® property will only appear in a **Save** dialog, because it must be read from the Creo® model itself. In the following example, the **Material** column displays the value of the Creo® **Material** property, and appears on the **Save** dialogs because its tag, creomaterial, appears in the `table.columns.checkin` preference:

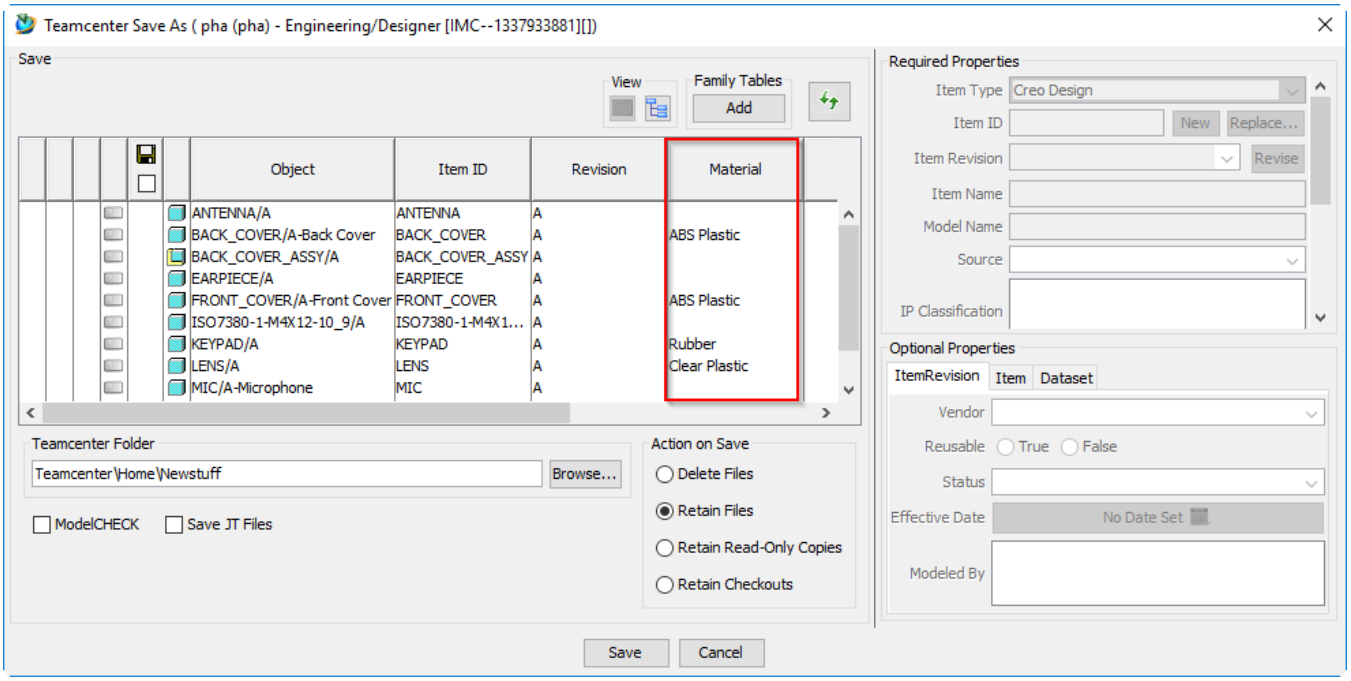


Figure 24 - Creo® property displayed on the Save dialog

A Creo® property displayed in a column does not require the map to be loaded into the Teamcenter database.

Column Map

Within a `column_map` definition can apear attribute definitions. Each attribute definition specifies the name of a property in the Creo® model (this is the attribute's *CAD name*), or the name of an attribute in Teamcenter (its *PDM name*). The attribute's CAD name or PDM name, but not both, must be specified in a column_map definition.

The `column_map` tells the integration about columns to be displayed in the dialogs. A `column_map` contains one or more `column` tags. Each column definition specifies a `tag` associated with a

preference value defined in the ipem.properties file. The `display_name` option tells the integration what to display in the column header. The `cad_name` option specifies the name of a property in the Creo® model (this is the attribute's CAD name) and the `pdm_name` option specifies the name of the corresponding attribute in Teamcenter (its PDM name).

column tag

```
<column tag="name">
```

The `column tag` specifies the name to be used in the preference value in the ipem.properties file. For example, if the name is "revdesc" then the name in ipem.properties must match and must be unique (independent of case) for the column map. Below is an example:

```
table.columns.std = itemid:itemtype:itemrev:itemname:revdesc:datasettype:cadname
```

display_name

```
<display_name value="Column Description"/>
```

The `display_name` tag specifies the column description.

cad_name

```
<cad_name value="name"/>
```

The `cad_name` tag specifies the property's name in the Creo® model, or it may be the name of an attribute function. The value appearing in the column will be the actual value from the Creo® property, not the display value from Teamcenter. If the `cad_name` tag is defined, then the `pdm_name` tag cannot also be defined. If it is the name of a property, it must obey the usual rules for Creo® property names. Only dialogs for operations that have access to the Creo® file, such as **Save**, should be configured with a `cad_name` column.

pdm_name

```
<pdm_name value="keyword"|"class[:type].name"/>
```

The `pdm_name` tag specifies the attribute's name in Teamcenter. The value appearing in the column will be the display value from Teamcenter. If the `pdm_name` tag is defined, then the `cad_name` tag cannot also be defined. This name can be a keyword for certain special attributes, or it can be a more general expression that gives the attribute's name and the class and type of object where the attribute can be found. Both the keyword and the general expression syntax are the same as documented for the `pdm_name` tag in the Attribute Mapping section of this document. Only dialogs for operations that access data from Teamcenter, such as **Open**, **Assemble Component**, **Insert UDF**, **Cache Manager** and **Cancel Checkouts**, should be configured with a `pdm_name` column.

Note:

Neither Teamcenter runtime properties nor reference properties can be used in a column map.

BOM Map

The `bom_map` tag defines the view type for the BOMs that the Integration must update each time it saves a Creo® model in Teamcenter. It has the following format.

```
<bom_map>
  <bom_line parent_type="parent" child_type="child" view_types="type">
  </bom_line>
      .
      .
      .
</bom_map>
```

BOM View Type

Each `bom_line` tag specifies the type of BOM view that must be created or updated whenever a Creo® model is saved. Both *parent* and *child* must be Creo® model types defined by entity tags in the `type_map`. The *parent* is the type of model for which the BOMs are being created. The *child* is the type of model that will appear on the BOM line. The *parent* is a model that depends on the *child*. The *type* is the type of BOM that should be given a BOM line for the child when the parent model is saved to Teamcenter.

For example, the following `bom_map` defines two BOM lines, one for assembly-to-assembly relationships, and one for assembly-to-part relationships. The first `bom_line` tag states that, when an assembly model (`parent_type`) depends on another assembly model (`child_type`), the item revision containing the parent model should be given a BOM of the `view` type. The second `bom_line` tag makes a similar statement about the BOM to add to an assembly item revision that depends on a part model.

```
<bom_map>
  <bom_line parent_type="ASM" child_type="ASM" view_types="view">
  </bom_line>
  <bom_line parent_type="ASM" child_type="PRT" view_types="view">
  </bom_line>
</bom_map>
```

Multiple view types are not supported. If multiple view types are defined, only the first view_type will be used. The `view_types` value is case sensitive. If the case is wrong or if the specified view type does not exist, save operations will fail with a message stating that “PS object not found in the database”.

BOM mapping for Part and Assembly MultiCAD reuse

Customers can open non-native Creo® models from Teamcenter, add them to their native Creo® assemblies, and save the native Creo® assembly to Teamcenter, retaining the non-native models as assembly components. Additional entries to the `bom_map` are needed to identify the eligible relationships between non-native parts and assemblies, and between non-native components and their native Creo® assemblies. In this example, showing the OOTB mapping, the integration is configured to

reuse 3D design data (both parts and assemblies) from SOLIDWORKS®, Inventor®, SolidEdge® and Catia®. NX® reuse is configured only at the part level; this is achieved with the `assembly="ignore"` tag, which is explained below in more detail.

```
<bom_map>
    <bom_line child_type="ASM" parent_type="ASM" view_types="view"/>
    <bom_line child_type="PRT" parent_type="ASM" view_types="view"/>
    <bom_line child_type="JT" parent_type="ASM" view_types="view"/>
    <bom_line child_type="SWASM" parent_type="ASM" view_types="view"/>
    <bom_line child_type="SWPRT" parent_type="ASM" view_types="view"/>
    <bom_line child_type="SWASM" parent_type="SWASM" view_types="view"/>
    <bom_line child_type="SWPRT" parent_type="SWASM" view_types="view"/>
    <bom_line child_type="AIASSEMBLY" parent_type="ASM" view_types="view"/>
    <bom_line child_type="AIPART" parent_type="ASM" view_types="view"/>
    <bom_line child_type="AIASSEMBLY" parent_type="AIASSEMBLY" view_types="view"/>
    <bom_line child_type="AIPART" parent_type="AIASSEMBLY" view_types="view"/>
    <bom_line child_type="SE ASSEMBLY" parent_type="ASM" view_types="view"/>
    <bom_line child_type="SE PART" parent_type="ASM" view_types="view"/>
    <bom_line child_type="SE ASSEMBLY" parent_type="SE ASSEMBLY" view_types="view"/>
    <bom_line child_type="SE PART" parent_type="SE ASSEMBLY" view_types="view"/>
    <bom_line child_type="CATPRODUCT" parent_type="ASM" view_types="view"/>
    <bom_line child_type="CATPART" parent_type="ASM" view_types="view"/>
    <bom_line child_type="CATPRODUCT" parent_type="CATPRODUCT" view_types="view"/>
    <bom_line child_type="CATPART" parent_type="CATPRODUCT" view_types="view"/>
    <bom_line child_type="UGMASTER" parent_type="ASM" view_types="view"/>
    <bom_line child_type="UGMASTER" parent_type="UGMASTER" view_types="view" assembly="ignore"/>
</bom_map>
```

The syntax is the same as is used for native Creo® assembly relations, except that non-native assemblies are not eligible for occurrence-level attribute mapping via the `bom_prop` tag.

Note:

Only part-level JT files may be reused – attempting to insert or open an assembly-level JT from Teamcenter will result in a Creo® error. In addition, JT datasets must be attached to their item revisions with the **IMAN_Rendering** or **IMAN_specification** relation.

BOM mapping for Part-only MultiCAD reuse

Certain types of non-native data can be directly reused by Creo® only at the part level. This is always the case for JT (DirectModel) and NX® files. With JT, there is no assembly file to open in Creo®. With NX® managed in Teamcenter, assemblies can only be opened by the NX® integration itself. For both of these data types, the integration can be configured to dynamically construct placeholder assemblies in the Creo® session, using the part files downloaded from Teamcenter and positioning them with information read from the assemblies' Bills of Materials. The placeholder assemblies are transient – they exist only the Creo® session, and cannot be saved back to Teamcenter.

Note:

A global option in the ipem.properties file, **checkout.dic.enable**, must retain its OOTB **true** value in order for the integration to open JT and NX® assemblies as described in this section. Do not modify this preference's value unless you have a specific reason for doing so.

In the **bom_map** example above, the entries for the **UGMASTER** dataset type have been given a special extension: `assembly="ignore"`. This tells the integration to ignore assembly-level **UGMASTER** datasets and download only the parts, using them to dynamically construct each NX® assembly as a transient Creo® placeholder. This extension is not needed for the JT dataset type, due

to legacy implementation details, but it could be applied to any other supported non-native CAD type. However, there is little reason for doing so, if the non-native assemblies can be directly opened into Creo®.

Note:

Adding `assembly="ignore"` to the `bom_line` tag is only valid if **checkout.dic.enable** = **true**.

BOM Properties

A `bom_line` tag may optionally enclose several `bom_prop` tags, each of which specifies a PTC feature parameter to be synchronized with an occurrence property on the Teamcenter BOM line. The `bom_prop` tag contains a direction for the mapping. Any PTC feature parameter and any Teamcenter BOM line property can be supported. Note that not all Teamcenter BOM line properties can be modified. The `bom_prop` tag can be repeated to establish multiple mappings. Each Teamcenter BOM line property added to the mappings must also be added to the `ipem_bomattr_policy.xml` file in the integration installation folder (%IPEM_DIR%\policies)

To enable BOM line attribute mapping, add the `bom_prop` tag to the appropriate entries in the `bom_map`, as shown in this example:

```
<bom_map>
  <bom_line parent_type="ASM" child_type="ASM" view_types="view">
      <bom_prop cadName="FindNo" pdmName="bl_sequence_no" direction="pdmtocad"/>
  </bom_line>
  <bom_line parent_type="ASM" child_type="PRT" view_types="view">
      <bom_prop cadName="FindNo" pdmName="bl_sequence_no" direction="pdmtocad"/>
  </bom_line>
</bom_map>
```

This configuration specifies that the value of the Teamcenter **Find No.** property will be mapped to a PTC feature parameter called **FindNo**. You must use the database name of the Teamcenter property, not its displayable name. The database name of the **Find No.** property is **bl_sequence_no**, so that value is used in the `bom_prop` entry. In the example above, `direction` is specified as `pdmtocad`, which means the mapping is uni-directional, from Teamcenter to Creo®. Like model parameter mappings, the direction can be `cadtopdm`, `pdmtocad`, or `both`.

Note

Feature parameters are different from model parameters in that they can be set on each of multiple occurrences of a part within the assembly model tree.

When an assembly is opened from Teamcenter, using the `bom_map` example shown above, the **FindNo** feature parameter of each first-level occurrence in the Creo® model tree is synchronized to match the **Find No.** property in the assembly BOM. If the Creo® feature parameter does not already exist for one or more occurrences, it will be created by the integration:

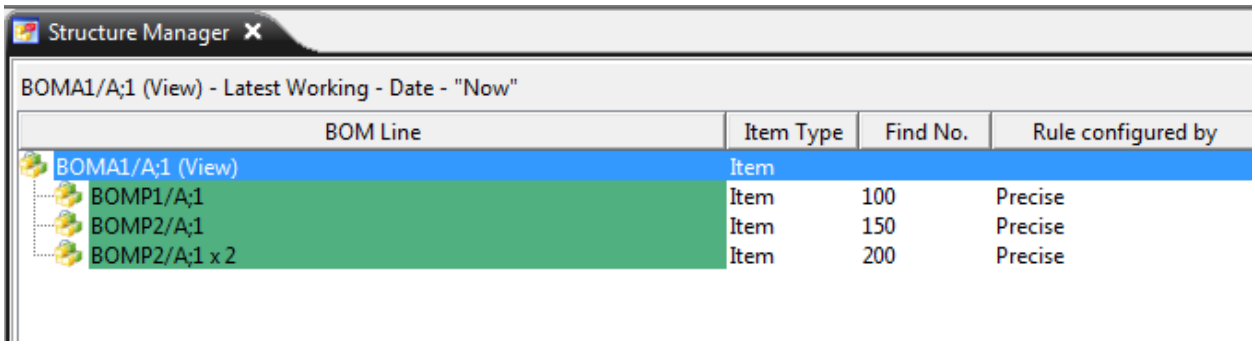


Figure 25 - Find Numbers in Teamcenter

Note

In the OOTB configuration, BOM properties are mapped only to the first-level children of the top-level assembly. This can be changed using the **checkout.bom.attributemapping.onelevelonly** preference in ipem.properties, as described under Open preferences.

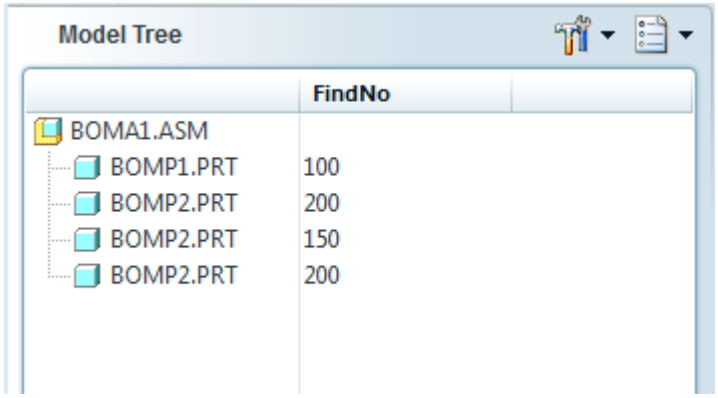


Figure 26 - Resulting feature parameters in Creo®

Configuring BOM Property Mappings

Every `bom_prop` tag has the following structure

```
<bom_prop cadName="FindNo" pdmName="bl_sequence_no" direction="pdmtocad"/>
```

`cadName`	The name of the PTC feature parameter.
`pdmName`	The database name (not display name) of the Teamcenter bomline property. For example `bl_sequence_no` is the name of the property displayed as **Find No**. All bomline properties are supported.
`direction`	Allowed values are `both`, `pdmtocad`, and `cadtopdm`. If `both`, then the property will be set in Teamcenter during save and in Creo® during open. If `pdmtocad`, then the property will be set in Creo® during open. If `cadtopdm`, then the property will be set in Teamcenter during save.

When `bom_prop` tags are defined in the ipem.xml file, cadtopdm synchronization occurs on every **Save**. Pdmtocad synchronization occurs on every **Update Attributes**, **Open**, and **Update Model** operation.

Note

If using BOM packing in the Structure Manager, the packing will be affected by pdmtocad mapping of bl_sequence_no. In some cases, portions of a BOM could become packed or unpacked unexpectedly.

When mapping in the cadtopdm direction, the BOM properties must be writeable. If not writeable, the save operation will fail. At a minimum, the BVR will be missing. The integration assumes write access to all mapped BOM line properties, as verification of the property on every BVR would have a significant performance impact.

String attributes being mapped in the cadtopdm direction will be truncated if the value in cad is greater than the maximum allowable length of the destination Teamcenter property.

If any BOMLine properties are configured as source or destination in the bom_prop tag of ipem.xml, those properties must be present in the `ipem_bomattr_policy.xml` file. If not already present, the site administrator should add the property under “ObjectType name=BOMLine”, as shown below. The policy files are in the integration’s installed client directory (%IPEM_DIR%\policies), on each Creo® workstation.

```
<ObjectType name="BOMLine">
        <Property name="Property1"/>
        <Property name="Property2"/>
         :
         :
        <Property name="Property"/>
</ObjectType>
```

Note:

Errors or performance degradation can occur if the BOMLine properties referenced as pdmName are not included in the policy file. Specifically, “assertion failure” errors may appear in the txdlog file if an attribute is missing from the policy file

Mappings to types other than Feature Parameters

Except for the specialized ECAD_REF_DES parameter, the destination for all BOM property mappings must be a Creo® assembly feature parameter. The following table may be extended to include more non-feature parameters, in future integration releases.

Parameter Type		Parameter Name	CadName in mapping
ECAD		ECAD_REF_DES	ECAD_REF_DES

Note:

Sites must not create a general Feature Parameter named ECAD_REF_DES.

Auxiliary Files

When a model is saved to Teamcenter, additional local files may be saved with it. These files may also be fetched from Teamcenter when the model is fetched. These additional files are called *auxiliary files*. Examples of auxiliary files are JT files for the Teamcenter embedded viewer, JPEG image files, and PDF files. The `auxiliary_file_map` tag tells the integration about auxiliary files to save or fetch with Creo® models.

An `auxiliary_file_map` tag contains one or more `auxiliary_file` tags. The general-purpose `auxiliary_file` tag specifies a pattern for the name of a file, the dataset type and named reference type where the file will be stored in Teamcenter, and optional commands for the operating system to execute. It may also specify a checkbox to display in the **Save** dialog for users to choose whether auxiliary files should be saved.

The auxiliary file map can be configured with any combination of the following settings:

- Save and retrieve existing files from/to the current workspace
    - In this mode, the auxiliary files must already exist in the current workspace, or in Teamcenter. See the section on **General Auxiliary Files**.
- Automatically export and save derived files during save to Teamcenter, using special-purpose, type-specific keywords
    - Several file types are given special handling, via keywords, such that they can be automatically created and saved to Teamcenter, along with their source Creo® files. See the section on **Specialized auxiliary files**.
- Automatically export and save derived files during save to Teamcenter, using OOTB or customized user exits
    - Any file type that can be exported from Creo®, using the **File→Save a Copy** function, can be automatically exported and saved to Teamcenter, along with its source Creo® file. The export is done with extensible and customizable Java code, invoked by a specialized auxiliary file tag, and working examples are provided OOTB. See the section on the **user_exit** tag.

Note:

This section documents the most commonly-used configuration settings and techniques. For a full description of all the available options, please refer to the **Auxiliary Files** section of the **Appendix**.

General Auxiliary Files

The `auxiliary_file` tag supports most kinds of auxiliary files. Each is associated with a particular type of Creo® model. When a model of that type is saved to Teamcenter, the Integration will also save any local files that match a specified file name pattern. When the model is fetched from Teamcenter, these files can also be fetched.

For example, here is an auxiliary file map containing a single `auxiliary_file` tag for **VRML** files. The `cad_type="ASM"` attribute means this auxiliary file definition is associated only with assembly models:

```
<auxiliary_file cad_type="ASM" direction="both">
      <pdm_location pdm_type="Pa4VRML" named_ref="Pa4vrml_ref"
      relation_type="IMAN_specification"/>
      <file_name pattern="{cad_name,lower}_asm.wrl"/>
      <cadtopdm_control label="Save VRML Files" user_preference_default="true"/>
      <os_command cmd="copy &quot;{cad_directory}\{cad_name,lower}_asm.wrl&quot;
      {temp_dir}" phase="in_directory"/>
</auxiliary_file>
```

The `direction` tag value is both, meaning that VRML files will be both saved to Teamcenter when the Creo® assembly is saved, and downloaded from Teamcenter when the Creo® assembly is opened. The integration does not provide a **type-specific keyword** for VRML auxiliary files, and this example does not use the **user_exit** tag, so the VRML file(s) must have been exported by the user prior to **Save**.

The `pdm_location` tag tells the Integration where to put the auxiliary file in Teamcenter, or where to retrieve it from. In the example above, the file will be stored in a dataset of type **Pa4VRML** and the file's named reference will be **Pa4vrml_ref**[3]. The dataset is assumed to have the same name as the Creo® model's dataset. If the dataset does not exist, it will be created and attached to the model's item revision with an **IMAN_specification** relation.

The `file_name` tag tells the Integration how to find the auxiliary file in the local file system. In the example above, the name of the auxiliary file is composed of the model's lower-case name with a **.wrl** extension. It is not an error if an auxiliary file cannot be found.

Note

If an error occurs while attempting to execute the command, the auxiliary file will not be saved to Teamcenter.

The `cadtopdm_control` tag defines an optional checkbox to display in the **Save** dialog's **Auxiliary Files** box. In this example, the checkbox label will be **Save VRML Files**. The `user_preference_default` tag value is **true**, meaning that the check box will be selected by default. If the `cadtopdm_control` tag is omitted, the check box will not appear on the dialog, and the specified auxiliary file will always be saved to Teamcenter, if found in the user's current workspace.

Finally, the `os_command` tag defines an operating system command that should be executed when the enclosing `auxiliary_file` command is processed. In this example, it simply copies the VRML file to the system %TEMP% directory. See the **os_command** section for more details.

Note

As demonstrated in the example above, if a path specification in an auxiliary file map may contain one or more space characters, then it must be bracketed by the `"` delimiter character.

[3] In some cases, Teamcenter will not have an OOTB dataset type that is suitable for the auxiliary file type being configured, and it may be necessary to create a new custom dataset type. This example includes such a custom dataset type.

The tags that can be used in a general `auxiliary_file` definition are described in more detail below.

auxiliary_file

```
<auxiliary_file cad_type="type" [direction ="both"|"cadtopdm"|"pdmtocad"]>
      optional elements
</auxiliary_file>
```

The `auxiliary_file` tag is the general-purpose definition of an auxiliary file. The `cad_type` attribute defines the type of Creo® model to which the `auxiliary_file` tag applies. The allowed `v`alues for *type* are the three-letter extensions that Creo® uses to identify its model files. For example, `cad_type="PRT"` means the auxiliary file is associated only with part models.

More than one Creo® model type may be given for `cad_type`. Use a colon to separate each three-letter value from the next. For example, to indicate that an `auxiliary_file` tag applies to parts, assemblies, and drawings, the attribute should be `cad_type="PRT:ASM:DRW"`.

The `direction` attribute indicates whether to save the file to Teamcenter when the Creo® model is saved, fetch it from Teamcenter when the model is fetched, or both. Its value may be one of the following strings (these strings must be enclosed in double quotes as shown).

`"both"`	The auxiliary file is saved to Teamcenter when the Creo® model is saved, and it is fetched from Teamcenter when the model is fetched.
`"cadtopdm"`	The auxiliary file is saved to Teamcenter when the Creo® model is saved but is not fetched with the model. This is the default.
`"pdmtocad"`	The auxiliary file is fetched from Teamcenter with the Creo® model, but it is not saved when the model is saved.

The *optional elements* that can be included in an `auxiliary_file` definition are described below.

pdm_location

```
<pdm_location named_ref="ref" [pdm_type="dataset"] [relation_type="relation"]/>
```

The `pdm_location` tag specifies where an auxiliary file should be saved in Teamcenter. Auxiliary files associated with a particular model are always saved in the same item revision as the model, but they can be stored either in the model's dataset, or in a dataset of their own.

The `named_ref` value must always be specified. The `pdm_type` attribute is the type of dataset in which to store the auxiliary file. The dataset is assumed to have the same name as the dataset holding the Creo® model. If the `pdm_type` is not specified, the auxiliary file is stored in the model's dataset.

The **Name** value on the **Dataset Properties** tab in the BMIDE template must be used for `pdm_type`. The value in the **Reference** column in the **References** tab must be used for `pdm_type`. For example, if using a custom dataset type defined as follows:

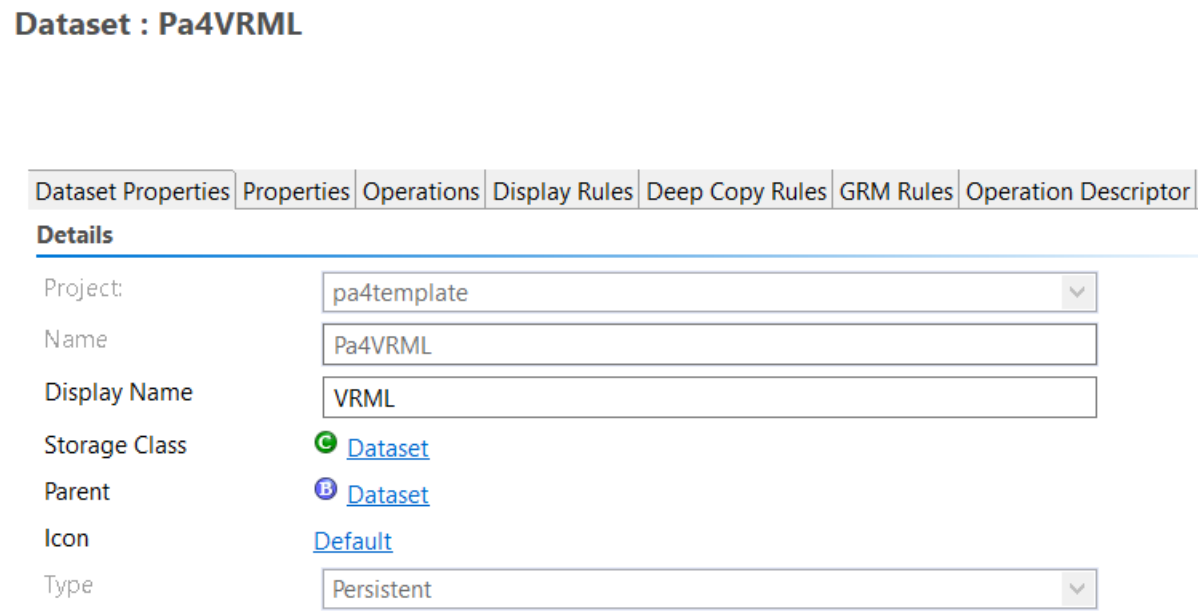


Figure 27 - The custom dataset's name

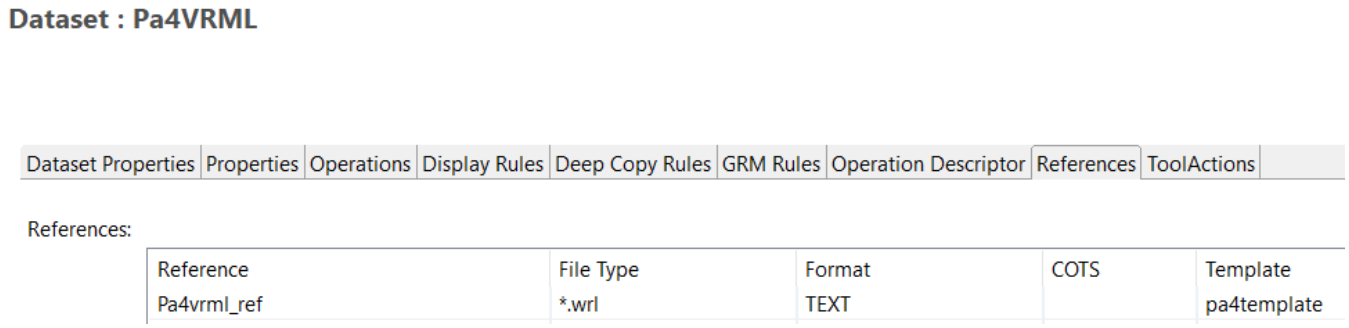


Figure 28 - The custom dataset's named references

Then the entry in the auxiliary map must be:

```
<pdm_location named_ref="Pa4vrml_ref" relation_type="IMAN_specification"
pdm_type="Pa4VRML"/>
```

If a dataset must be created for the auxiliary file, it will be attached to the item revision using the type of relation specified by `relation_type`. If not specified, the default relation type is **IMAN_Rendering**.

The `named_ref` attribute is required in a `pdm_location` definition. The `pdm_type` and `relation_type` attributes are optional.

file_name

```
<file_name pattern="pattern" [version="all"|"latest"]/>
```

The file_name tag identifies the auxiliary file by defining a pattern that matches the file's name. The pattern may include certain keywords that the Integration will replace with data from the model. Keywords can be used anywhere in a pattern, both in the file name part and the directory parts of a path. The recognized keywords are described later in the section on **Substitution Keywords**.

The * wildcard character is allowed, but only in the simple file name part of the pattern. For example, "myfile.*" and "{temp_dir}/*.gif" are acceptable patterns.

The optional version attribute indicates whether all auxiliary files matching the pattern should be saved to Teamcenter, or only the file with the latest modification time. The default is "all", which saves all auxiliary files that match the pattern.

The file_name tag is optional. If it is not included in an auxiliary_file tag, the Integration will not attempt to find and save an auxiliary file when it saves the corresponding Creo® model, but it will still execute any operating system commands specified by the os_command tag.

os_command

```
<os_command [cmd="command"] [pre_cmd="command"] [post_cmd="command"]
[phase="in_cad"|"in_directory"|"in_pdm"]
[ignore_status="true"|"false"]/>
```

The os_command tag specifies one, two, or three optional commands for the operating system to execute. By default, these commands are executed after Creo® has saved the models to the local file system, but before the models or auxiliary files are checked-in to Teamcenter.

Note

If an error occurs while attempting to execute one of these commands, the auxiliary files affected by that particular auxiliary_file tag will not be checked into Teamcenter.

The string in cmd="*command*" is an operating system command that is executed for each Creo® model saved to Teamcenter. The command string may include the same keywords allowed in the file_name pattern, which are listed in the section on **General Substitution Keywords**.

Note:

Literal paths, whether absolute or relative, are not supported, and will result in errors during **Save** to Teamcenter.

The strings in pre_cmd="*command*" and post_cmd="*command*" are also operating system commands. The pre_cmd string is executed once, before any cmd strings from any auxiliary_file tag are executed ("once" means just once during a **Save** or **Save All** operation, regardless of the number of models being saved). The post_cmd string is also executed only once, after all cmd strings from all auxiliary_file tags have been executed. The pre_cmd and post_cmd strings may also include some of the keywords described in the **General Substitution Keywords** section, but not every keyword is meaningful in these command strings. Since pre_cmd and post_cmd are not associated with one specific model, keywords that refer to a model's name, directory, item ID, or other model-specific data, are not applicable in these strings.

If you provide a pre_cmd, cmd, or post_cmd, the Integration requires the command to execute successfully before it saves the auxiliary file to Teamcenter. If pre_cmd, cmd, or post_cmd fails, the auxiliary file matched by the file_name pattern will not be saved. In addition, if pre_cmd fails, neither cmd nor post_cmd will be executed.

If an operating system command exits with a non-zero status, the Integration usually assumes the command has failed. However, if ignore_status="true" is specified, the Integration ignores exit status and assumes commands always succeed when executed.

You can also specify the *phase* in which these commands will execute. When the Integration saves models from Creo® to Teamcenter, the models pass through three phases corresponding to the values that can be assigned to the phase attribute:

in_cad	The models are in the Creo® session and have not yet been saved to the local file system.
in_directory	The models have been saved from Creo® to the local file system, but neither the models nor any auxiliary files have been checked in to Teamcenter.
in_pdm	All models and auxiliary files have been checked in to Teamcenter.

By default, the commands specified by an os_command tag are executed during the in_directory phase, which is usually the phase in which you would want to run any translators that read a Creo® file to create an auxiliary file. If you have any batch processes that must run after models have been checked in to Teamcenter, you might want to use an os_command tag to launch them during the in_pdm phase. As a point of interest, the cgm, dxf and plot tags do their work during the in_cad phase when Creo® models may be modified in session before being saved to the file system, and the ets_request tag submits translation requests during the in_pdm phase.

You can think of the os_command, create_text_file, cgm, dxf, ets_request and plot tags as actions within an auxiliary_file definition. Each auxiliary_file definition may have one such action per phase, which means that as many as three os_command tags can appear in an auxiliary_file definition if each one specifies a different phase. Alternatively, an auxiliary_file definition could have one dxf or plot tag, which acts during the in_cad phase, a create_text_file tag for the in_directory phase, an os_command tag for the in_pdm phase, and so on. For example, an auxiliary_file definition could have a plot tag to make Creo® generate a PostScript® file, and an os_command tag to convert it to a PDF document during the in_directory phase.

Every attribute in an os_command tag is optional.

cadtopdm_control

```
<cadtopdm_control label="label" [user_preference_name="name"]
[user_preference_default="true"|"false"]/>
```

The cadtopdm_control tag creates a checkbox in the **Save** dialog that allows the user to choose whether to save the auxiliary files specified by the auxiliary_file definition. The text labeling the checkbox is *label*. If the same label is used for several auxiliary_file definitions, a single checkbox can control all of them.

The user_preference_name attribute creates a user preference *name* that can be used in the ipem.properties file to set the initial state of the checkbox. When the user preference is not set, the

checkbox is initially checked or unchecked according to whether `user_preference_default` is true or false.

If an `auxiliary_file` definition has no `cadtopdm_control` tag, the auxiliary files are always saved.

Note

The `direction` attribute on the `auxiliary_file` definition must be `"cadtopdm"` or `"both"` for the `cadtopdm_control` tag to have any effect.

Creo® Accelerator files for SOLIDWORKS® MultiCAD

As another example of general auxiliary files, SOLIDWORKS® requires the Creo® accelerator (.xas and .xpr) files to open Creo® family table instances into the SOLIDWORKS® session. The Creo® integration must be configured to store the accelerator files in Teamcenter, for use by SOLIDWORKS®.

```
<auxiliary_file cad_type="ASM" direction="both">
    <pdm_location named_ref="AccelFile"/>
    <file_name pattern="{cad_name,lower}.xas"/>
</auxiliary_file>
<auxiliary_file cad_type="PRT" direction="both">
    <pdm_location named_ref="AccelFile"/>
    <file_name pattern="{cad_name,lower}.xpr"/>
</auxiliary_file>
```

Note

The above mapping of Creo® accelerator files is configured OOTB – it must be retained if Creo® models are to be reused in SOLIDWORKS®.

Specialized auxiliary files

The integration supports certain auxiliary file types with specialized tags, for convenience.

JT Files

The `jt_file` tag is simpler than the `auxiliary_file` tag for configuring the integration to generate and save JT files.[4] The `jt_file` element shown below in the `auxiliary_file_map` makes this happen:

```
<auxiliary_file_map>
  <jt_file action="translate" cad_type="PRT"/>
</auxiliary_file_map>
```

This tells the integration to generate and save a JT file with each Creo® part model saved to Teamcenter (`cad_type="PRT"`). To save JT files for both parts and assemblies, use `"PRT:ASM"` for the `cad_type` value.

Additional attributes of the `jt_file` tag can separately control whether JT files are generated and whether they are saved. For example, the following `jt_file` element tells the integration that it should

[4] Siemens' **PLM Components JT Translator for Creo®** must be installed to generate JT files.

not generate JT files (`action="none"`), but it should nonetheless save JT files (`save="true"`) that have the same name and are in the same directory as a model being saved to Teamcenter. This may be preferred by users who want to generate their JT files with the interactive EAI translator in Creo®.

```
<auxiliary_file_map>
  <jt_file cad_type="PRT" action="none" save="true"></jt_file>
</auxiliary_file_map>
```

cgm

```
<cgm [sheets="all"|"current"]
       [file_structure="cleartext"|"milspec"]
       [coordinates="abstract"|"metric"]>

      <pro_config name="config_option" value="config_value"/>

      <pro_layer name="layer" status="show"|"blank"|"hidden"/>

</cgm>
```

The `cgm` tag configures Creo® to create a file in the Computer Graphics Metafile (CGM) format, from a drawing. The result is similar to using Creo®'s **File→Save a Copy** command to create the CGM file. By default, a separate CGM file is created for each sheet in the drawing, in the same directory as the drawing. The CGM file is given the same name as the drawing with the sheet number appended, followed by a `.cgm` extension. For example, if a drawing named WIDGET has three sheets, three files will be created named `widget1.cgm`, `widget2.cgm`, and `widget3.cgm`.

The optional `sheets` attribute determines whether a CGM file is created for each sheet in the drawing, or only for the current sheet. If `sheets` is `"current"`, a single CGM file is created. In addition, when `sheets` is `"current"`, the sheet number is not included in the CGM file name.

Note

The *`sheets`* attribute is ignored when using a Creo® version prior to Wildfire. Only a single CGM file for the current sheet will be created if an earlier Creo® version is used.

The `file_structure` and `coordinates` attributes specify the same options that Creo® would prompt you to enter if you were to use the **File | Save a Copy** command to create the CGM file. If `file_structure` is `"cleartext"`, the CGM file will be a text file. If `file_structure` is `"milspec"`, the CGM file will be a binary file. The `coordinates` attribute indicates whether the data in the CGM file should be defined in abstract or metric units.

An auxiliary file definition that uses `cgm` to create a CGM file and save it to an Image dataset whenever a drawing is saved to Teamcenter might look like this:

```
<auxiliary_file cad_type="DRW">
  <pdm_location pdm_type="Image" named_ref="Image"/>
  <file_name pattern="{cad_name,lower}*.cgm"/>
  <cadtopdm_control label="Save CGM Files"/>
  <cgm sheets="current"/>
</auxiliary_file>
```

In this example, the `cadtopdm_control` tag adds a checkbox to the **Save** dialog, giving the user control over whether the CGM file will be created. Notice also that a separate `file_name` tag is needed to tell the Integration where to find the file after it is created, and the `pdm_location` tag tells it where to put the file in Teamcenter.

Two sub-tags, `pro_config` and `pro_layer`, can augment the `cgm` tag with additional changes to Creo®'s configuration. Refer to the `plot` tag for details on using the `pro_config` and `pro_layer` tags.

The `cgm` tag always creates CGM files during the `in_cad` phase. An `auxiliary_file` tag should not have more than one action per phase, so do not use other `in_cad` actions within the same `auxiliary_file` tag as your `cgm` tag. See the section on the `os_command` tag for more information on phases.

dxf

```
<dxf [sheets="all"|"current"]>
<pro_config name="config_option" value="config_value"/>
                                    ⋮
<pro_layer name="layer" status="show"|"blank"|"hidden"/>

</dxf>
```

The `dxf` tag configures Creo® to create a DXF™ file from a drawing. The result is similar to using Creo®'s **File | Save a Copy** command. By default, a separate DXF file is created for each sheet in the drawing, in the same directory as the drawing. The DXF file is given the same name as the drawing with the sheet number appended, followed by a `.dxf` extension. For example, if a drawing named WIDGET has three sheets, three files will be created named `widget1.dxf`, `widget2.dxf`, and `widget3.dxf`.

The optional `sheets` attribute determines whether a DXF file is created for each sheet in the drawing, or only for the current sheet. If `sheets` is `"current"`, a single DXF file is created. Also, when `sheets` is `"current"`, the sheet number is not included in the DXF file name.

Note

The *`sheets`* attribute is ignored when using a Creo® version prior to Wildfire. Only a single DXF file for the current sheet will be created if an earlier Creo® version is used.

An auxiliary file definition that uses `dxf` to create a DXF file and save it to an DXF dataset whenever a drawing is saved to Teamcenter might look like this:

```
<auxiliary_file cad_type="DRW">
  <pdm_location pdm_type="DXF" named_ref="DXF"/>
  <file_name pattern="{cad_name,lower}*.dxf"/>
  <cadtopdm_control label="Save DXF Files"/>
  <dxf sheets="current"/>
</auxiliary_file>
```

In this example, the `cadtopdm_control` tag adds a checkbox to the Save dialog, giving the user control over whether the DXF file will be created. Notice also that a separate `file_name` tag is needed to tell the Integration where to find the file after it is created, and the `pdm_location` tag tells it where to put the file in Teamcenter.

Two sub-tags, `pro_config` and `pro_layer`, can augment the `dxf` tag with additional changes to Creo®'s configuration. Refer to the `plot` tag for details on using the `pro_config` and `pro_layer` tags.

The `dxf` tag always creates DXF files during the `in_cad` phase. An `auxiliary_file` tag should not have more than one action per phase, so do not use other `in_cad` actions within the same `auxiliary_file` tag as your `dxf` tag. See the section on the `os_command` tag for more information on phases.

ets_request

```
<ets_request translator="service" [provider="name"] [priority="0"|"1"|"2"|"3"]
[request_per_model="true"|"false"]>

  ⋮
</ets_request>
```

The `ets_request` tag submits translation requests to Teamcenter's Dispatcher Framework after all models have been checked in to Teamcenter. The name of the translator that will handle the request, also known as the *service*, is given by the required `translator` attribute. The default provider is `"SIEMENS"`, but other providers can be specified with the optional `provider` attribute. The translation request can be given a priority of 1, 2 or 3, corresponding to low, medium, or high priority, using the `priority` attribute. If the priority is set to 0, no translation request is submitted.

By default, the `ets_request` tag submits a separate translation request for each Creo® model of the specified type that is saved to Teamcenter. However, if the translator can handle more than one model at a time, you can set the `request_per_model` attribute to `"false"`, which will cause one translation request to be submitted containing all models that have been saved to Teamcenter. For example, if four parts are saved to Teamcenter and an `auxiliary_file` definition with a `cad_type` of `"PRT"` specifies an `ets_request`, four separate translation requests will be submitted by default or if `request_per_model` is explicitly `"true"`. If `request_per_model` is `"false"`, a single translation request will be submitted containing all four parts.

If an auxiliary file definition uses `ets_request`, ETS will create the auxiliary file and import it into Teamcenter, so there is no need to specify a file or a PDM location for it. For example, an auxiliary file definition that submits a request for translating drawings to DXF files might look like this:

```
<auxiliary_file cad_type="DRW">
  <cadtopdm_control label="Save DXF Files (ETS)"/>
  <ets_request translator="proetodxf"/>
</auxiliary_file>
```

In this example, the `cadtopdm_control` tag adds a checkbox to the Save dialog, giving the user control over whether the DXF file will be created.

The `ets_request` tag always submits ETS requests during the `in_pdm` phase. An `auxiliary_file` tag should not have more than one action per phase, so do not use other `in_pdm` actions within the same `auxiliary_file` tag as your `ets_request` tag. See the section on the `os_command` tag for more information on phases.

When `<ets_request translator="proetojt"/>`, translation request for assemblies are created based on the Teamcenter preference `ETS.ASSEMBLYTYPES.SIEMENS.PROETOJT`. See the **Teamcenter Preferences** section for details.

Note

Creo® skeleton models do not generate ETS requests for JT translation.

pdf

```
<pdf [sheets="all"|"current"][create_separate_files="true"|"false"]>
<pro_config name="config_option" value="config_value"/>
                                        ⋮
<pro_layer name="layer" status="show"|"blank"|"hidden"/>
                                        ⋮
</pdf>
```

The `pdf` tag configures Creo® to create a PDF file from a drawing. The result is similar to using Creo®'s **File | Save a Copy** command. By default, a separate PDF file is created for each sheet in the drawing, in the same directory as the drawing. The PDF file is given the same name as the drawing with the sheet number appended, followed by a `.pdf` extension. For example, if a drawing named WIDGET has three sheets, three files will be created named `widget1.pdf`, `widget2.pdf`, and `widget3.pdf`.

The optional `sheets` attribute determines whether a PDF file is created for each sheet in the drawing, or only for the current sheet. If `sheets` is `"current"`, a single PDF file is created. In addition, when `sheets` is `"current"`, the sheet number is not included in the PDF file name.

The optional `create_separate_files` attribute specifies if a separate PDF file should be created for each sheet in a drawing or layout (`"true"`), or all sheets should be plotted to a single file (`"false"`).

If the `create_separate_files` attribute is `"true"`, Creo® will automatically add a sheet number to the file name when a drawing or layout is plotted, inserting the sheet number between the file's base name and its extension. For example, if `file_name="{cad_name,lower}.pdf"` is given, then Creo® will create files `widget1.pdf`, `widget2.pdf`, and `widget3.pdf` when plotting a 3 sheet drawing named widget.

Note

If you choose to create separate files for each sheet in a drawing or layout, the *pattern* attribute of the *file_name* tag should include a wildcard for the added sheet number, such as *<file_name pattern="{cad_name,lower}*.pdf"/>* for the example above. Note that it is the *pattern* attribute of the *file_name* tag that gets the wildcard, not the *file_name* attribute of the *pdf* tag.

An auxiliary file definition that uses `pdf` to create a PDF file and save it to an PDF dataset whenever a drawing is saved to Teamcenter might look like this:

```
<auxiliary_file cad_type="DRW">
  <pdm_location pdm_type="PDF" named_ref="PDF_Reference"/>
  <file_name pattern="{cad_name,lower}*.pdf"/>
  <cadtopdm_control label="Save PDF Files"/>
  <pdf sheets="current"/>
</auxiliary_file>
```

In this example, the `cadtopdm_control` tag adds a checkbox to the **Save** dialog, giving the user control over whether the PDF file will be created. Notice also that a separate `file_name` tag is needed to tell the Integration where to find the file after it is created, and the `pdm_location` tag tells it where to put the file in Teamcenter.

Two sub-tags, `pro_config` and `pro_layer`, can augment the `pdf` tag with additional changes to Creo®'s configuration. Refer to the `plot` tag for details on using the `pro_config` and `pro_layer` tags.

The `pdf` tag always creates PDF files during the `in_cad` phase. An `auxiliary_file` tag should not have more than one action per phase, so do not use other `in_cad` actions within the same `auxiliary_file` tag as your `pdf` tag. See the section on the `os_command` tag for more information on phases.

plot

```
<plot
      plotter_name="plotter"
      [sheets="all"|"current"|"range" [first_sheet="sheet" last_sheet="sheet"]]
      [create_separate_files="true"|"false"]
      [scale="scale"]
      [label="true"|"false"]
      [interface_quality="0"|"1"|"2"|"3"]
      [paper_size="size"|"variable" [paper_size_x="length" paper_size_y="length"]]
      [segmented="true"|"false"]
      [pen_slew="true"|"false"|"rate" [pen_slew_x="rate" pen_slew_y="rate"]]
>
      <pro_config name="config_option" value="config_value"/>
                                          ⋮
      <pro_layer name="layer" status="show"|"blank"|"hidden"/>
                                          ⋮
</plot>
```

The `plot` tag configures Creo® to create a plotter file. Two attributes are required: the plotter type (`plotter_name`) and the file name for the plot that will be created (`file_name`). All other attributes are optional. Most of these attributes are the same ones that can be set when creating a plot with Creo®'s **File→Print** command, or in a Creo® plotter configuration file (also known as a `.pcf` file). If an attribute is not specified, its value is taken from Creo®'s current configuration when the plot is created.

Note

The plotter type can be determined from Creo®'s Printer Configuration dialog. In Creo®, choose **File→Print** and select the desired printer from the drop-down list. The plotter type will be shown at the top of the Printer Configuration dialog in the Type field.

An auxiliary file definition that uses `plot` to create a PostScript® file and save it to an Image dataset whenever a drawing is saved to Teamcenter might look like this:

```
<auxiliary_file cad_type="DRW">
  <pdm_location pdm_type="Image" named_ref="Image"/>
  <file_name pattern="{cad_name,lower}.ps"/>
  <cadtopdm_control label="Save PostScript Files"/>
  <plot plotter_name="COLORPOSTSC" file_name="{cad_name,lower}.ps"/>
</auxiliary_file>
```

In this example, the `cadtopdm_control` tag adds a checkbox to the **Save** dialog, giving the user control over whether the PostScript file will be created. Notice also that a separate `file_name` tag is needed to tell the Integration where to find the file after it is created, and the `pdm_location` tag tells it where to put the file in Teamcenter.

Two sub-tags can augment the `plot` attributes with additional changes to Creo®'s configuration. The `pro_config` sub-tag sets one or more Creo® configuration options that will be applied temporarily, while the plotter file is being created, after which the session's current configuration is restored. Each

pro_config sub-tag specifies a configuration option name *config_option* just as it would appear in a config.pro file, and a value *config_value* to assign to that option. Similarly, the pro_layer sub-tag temporarily shows or blanks one or more layers while the plotter file is being created. For example, to guarantee datums and axes are not displayed when creating a CGM file, the plot command might be configured this way.

```
<plot plotter_name="CGM" file_name="{cad_name,lower}.cgm">
  <pro_config name="AXIS_DISPLAY" value="NO"/>
  <pro_config name="DATUM_DISPLAY" value="NO"/>
  <pro_config name="DATUM_POINT_DISPLAY" value="NO"/>
</plot>
```

The plot tag's attributes are described in detail below.

create_separate_files="true"|"false"
: Indicates whether a separate plot file should be created for each sheet in a drawing or layout ("true"), or all sheets should be plotted to a single file ("false"). Note that Creo® adds a sheet number to the plot file's base name when a separate file is created for each sheet. Set this value to "true", if having problems printing multi-site drawings.

  If the create_separate_files attribute is "true", Creo® will automatically add a sheet number to the file name when a drawing or layout is plotted, inserting the sheet number between the file's base name and its extension. For example, if file_name="{cad_name,lower}.ps" is given, then Creo® will create files widget1.ps, widget2.ps, and widget3.ps when plotting a 3 sheet drawing named WIDGET.

Note

If you choose to create separate files for each sheet in a drawing or layout, the pattern attribute of the file_name tag should include a wildcard for the added sheet number, such as <file_name pattern="{cad_name,lower}*.ps"/> for the example above. Note that it is the pattern attribute of the file_name tag that gets the wildcard, not the file_name attribute of the plot tag.

first_sheet="*sheet*"
: Specifies the number of the first sheet to plot in a drawing or layout when sheets="range" is given. The sheet number must be a positive integer.

interface_quality="0"|"1"|"2"|"3"
: Controls output quality on a pen plotter by limiting the number of pen changes and overlapping lines. The highest quality is "3", which causes Creo® to test every edge for overlap and to organize all lines of the same color so they can be plotted without unnecessary pen changes. When the quality is "2", only edges with two vertices are tested for overlap, but lines of the same color are still organized to plot without unnecessary pen changes. A quality of "1" organizes lines of the same color but turns off all tests for overlapping edges.

The lowest quality is "0", which does not test for overlapping edges nor organize lines of the same color.

label="true"|"false"
Indicates whether to include labels on the plot.

last_sheet="*sheet*" Specifies the number of the last sheet to plot in a drawing or layout when sheets="range" is given. The sheet number must be a positive integer.

pen_slew="true"|"false"|"*rate*"
Specifies the pen slew rate. If "true", the slew rate is given by the pen_slew_x and pen_slew_y attributes. Specifying a number for *rate* is the same as setting pen_slew to "true" and assigning the same rate to pen_slew_x and pen_slew_y. If "false", slewing is not allowed.

pen_slew_x="*rate*" Specifies the pen slew rate in the x direction, where *rate* is a positive number. This attribute is used only when pen_slew="true" is given.

pen_slew_y="*rate*" Specifies the pen slew rate in the y direction, where *rate* is a positive number. This attribute is used only when pen_slew="true" is given.

plotter_name="*plotter*"
Specifies the plotter driver to use for creating the plot file. This is the printer type shown when the **Configure** button is clicked in Creo®'s **File→Print** dialog, not the name of the printer, and it implicitly defines the type of file to create. This attribute is required and has no default. Several common values for "*plotter*" are listed below (some are available only for certain Creo® model types), but a full list of the available printer types can be obtained using Creo®'s **File→Print→Add Printer** command.

Use This Plotter…	**To Create This Type of Plot…**
"CGM"	CGM
"HP7440A"	HPGL
"DesignJet"	HPGL 2
"JPEG"	JPEG
"postscript"	PostScript
"TIFF"	TIFF

scale="*scale*" A non-zero positive scale factor for the plot. The default is 1.

segmented="true"|"false"
Indicates whether a plot can be printed across several pages if the page size is smaller than the plot size ("true"), or printed on a single page ("false").

sheets="all"|"current"|"range"
Indicates which sheets to plot in drawings and layouts, where "all" plots all sheets, "current" plots only the current sheet, and "range" indicates a range of sheets bounded by the first_sheet and last_sheet attributes.

preview_command

```
<preview_command [cmd="command"] [pre_cmd="command"] [post_cmd="command"]
      [phase="in_cad"|"in_directory"|"in_pdm"]
      [ignore_status="true"|"false"]/>
```

The `preview_command` tag is used to control the generation and saving of preview images to Teamcenter. The tag specifies one, two, or three optional commands for the Creo® integration to execute. These commands are executed after the integration has saved the models to the local file system, but before the models or auxiliary files are checked in to Teamcenter. The `preview_command` tag is similar to the `os_command` tag except the keyword `built_in` is added to invoke additional integration functionality. The preview_command deletes the saved previews from disk once they are stored to Teamcenter. By default the `ipem.xml` file includes these preview command tags:

```
<auxiliary_file_map>
      <auxiliary_file cad_type="PRT" direction="cadtopdm">
            <!-- Uncomment the below if you want to give the user choice of saving the
            preview files -->
            <!-- cadtopdm_control label="Save Preview Files"/ -->
            <pdm_location named_ref="JPEG" pdm_type="ProPrt"/>
            <file_name pattern="{cad_directory}\{cad_name,lower}prt.JPG"/><preview_command
            cmd="built_in  {cad_directory}\{cad_name,lower}prt.JPG" ignore_status="true"
            phase="in_directory"/>
      </auxiliary_file>

      <auxiliary_file cad_type="ASM" direction="cadtopdm">
            <pdm_location named_ref="JPEG" pdm_type="ProAsm"/>
            <file_name pattern="{cad_directory}\{cad_name,lower}asm.JPG"/>
            <preview_command cmd="built_in {cad_directory}\{cad_name,lower}asm.JPG"
            ignore_status="true" phase="in_directory"/>
      </auxiliary_file>

      <auxiliary_file cad_type="DRW" direction="cadtopdm">
            <pdm_location named_ref="JPEG" pdm_type="ProDrw"/>
            <file_name pattern="{temp_dir}\{cad_name}d.JPG"/>
            <preview_command cmd="built_in {temp_dir}\{cad_name}d.JPG" ignore_status="true"
            phase="in_directory"/>
      </auxiliary_file>
</auxiliary_file_map>
```

Note

Do not change any of the existing the `cad_type`, `pdm_location`, `pdm_type`, `file_name pattern`, or `preview_command` tags in the OOTB auxiliary file map. Do not configure more than one type of preview auxiliary file for each unique CAD type.

The commented `cadtopdm_control label` is provided as an example of how to configure check boxes to appear on the save dialog, giving Creo® designers control over when to create the preview images, and for which CAD types.

Using the preview_command to display SOLIDWORKS preview images

If configured appropriately, the Creo® integration enables direct reuse of SOLIDWORKS®, or other, mechanical CAD data within the Creo® session. The following auxiliary file maps configure the integration to display the SOLIDWORKS® preview images in the Creo® integration dialogs; similar settings may be used to enable previews for other CAD types.

```
<auxiliary_file cad_type="SWPRT" direction="pdmtocad">
      <pdm_location named_ref="PNG" pdm_type="SWPrt"/>
      <preview_command cmd="built_in {cad_directory}/{cad_name,lower}sldprt.PNG"
      ignore_status="true" phase="in_directory"/>
</auxiliary_file>

<auxiliary_file cad_type="SWASM" direction="pdmtocad">
      <pdm_location named_ref="PNG" pdm_type="SWAsm"/>
      <preview_command cmd="built_in {cad_directory}/{cad_name,lower}sldasm.PNG"
      ignore_status="true" phase="in_directory"/>
</auxiliary_file>
```

user_exit

<user_exit file_name="*filename.ext*" [ignore_status="true"|"**false**"]/>

The `user_exit` tag configures Creo® to create a file of `filename.ext`, where `.ext` is the file name extension of the desired auxiliary file type. The translation is implemented by the `IpemUserExit.dll` found in `{ipem_dir}\bin` folder, on each integration client. The OOTB set of file types built into the `IpemUserExit.dll` include IGES, 3D PDF, Stereolithography (STL), STEP (STP) and TIFF. To customize those translations or and new capabilities modify the `IpemUserExit.dll`, by following the instructions in the **VC++ Customization points** section in this document.

This entry will cause the integration to translate and save STEP files for Creo® parts and assemblies, using the precompiled `IpemUserExit.dll` file, which is automatically loaded at runtime:

```
<auxiliary_file cad_type="ASM:PRT" direction="cadtopdm">
<pdm_location named_ref="Image" relation_type="IMAN_Rendering" pdm_type="Image"/>
<file_name pattern="{cad_name,lower}.stp"/>
<cadtopdm_control label="STEP" user_preference_default="false"/>
<user_exit file_name="{cad_name,lower}.stp" ignore_status="true"/>
</auxiliary_file>
```

Note

If you choose to create multiple files with the user exit tag, the *`pattern`* attribute of the *`file_name`* tag should include a wildcard for the added sheet number, such as *`<file_name pattern="{cad_name,lower}*.stp"/>`* for the example above. Note that it is the *`pattern`* attribute of the *`file_name`* tag that gets the wildcard, not the *`file_name`* attribute of the *`user_exit`* tag.

General Substitution Keywords

Patterns of the form {*keyword*} can be used in the `file_name` pattern, in the `os_command` and `create_text_file` tags, and in the `file_name` attribute of the `plot` tag, `cgm` tag and `dxf` tag. The Integration replaces these patterns with appropriate values. For example, if a model is named WIDGET, the pattern `"{cad_name}.gif"` will become `"WIDGET.gif"` when WIDGET is saved to Teamcenter.

The modifier `upper` or `lower` may be appended to any keyword. Use a comma to separate the modifier from the keyword, without spaces. The {*keyword*`,upper`} pattern will be replaced with a result that is all upper case. The {*keyword*`,lower`} pattern converts the result to lower case. For example, the pattern `"{cad_name,lower}.gif"` would produce `"widget.gif"` when the model's name is WIDGET.

The `escape` modifier can also be appended to any keyword. This modifier causes every backslash in the result to be replaced with two backslashes. For example, if the `{cad_directory}` pattern produces the result `"C:\work\latest"`, the pattern `{cad_directory,escape}` will produce `"C:\\work\\latest"`.

The recognized keywords are:

`{cad_dataset_uid}` The UID for the model's Teamcenter dataset, or an empty string if the dataset does not exist at the time this pattern is replaced. When this pattern is used in an `os_command` string, for example, the dataset might not exist during the `in_cad` and `in_directory` phases.

`{cad_directory}` The absolute path to the directory containing the model file.†

`{cad_file}` The absolute path to the model file.

`{cad_name}` The model's name.

`{cad_type}` The Creo® three-letter model type, such as ASM or PRT.

`{group}` The user's current Teamcenter group.

`{ipem_dir}` The absolute path to the directory where the Teamcenter Integration for Creo® is installed.

`{item_id}` The Teamcenter item ID for the model.

`{item_name}` The Teamcenter item name for the model.

`{item_revision_id}` The Teamcenter item revision ID for the model.

`{item_revision_uid}` The UID for the model's Teamcenter item revision, or an empty string if the item revision does not exist at the time this pattern is replaced. When this pattern is used in an `os_command` string, for example, the item revision might not exist during the `in_cad` and `in_directory` phases.

`{item_uid}` The UID for the model's Teamcenter item, or an empty string if the item does not exist at the time this pattern is replaced. When this pattern is used in an `os_command` string, for example, the item might not exist during the `in_cad` and `in_directory` phases.

† This keyword can be used only where a single model is implied, such as in the `file_name` pattern or in the `os_command cmd` string. It is not recognized in the `os_command pre_cmd` or `post_cmd` strings because these commands are not executed for each model.

`{named_ref}`	The named reference for the auxiliary file.
`{pdm_type}`	The Teamcenter dataset type for the auxiliary file. For example, if the auxiliary file is saved in the same dataset as a Creo® assembly model, the dataset type is ProAsm. For JT files, the dataset type is DirectModel.
`{proe_command}`	The command that starts Creo®.
`{proe_common_dir}`	The absolute path to the directory where Creo® common directory is installed (e.g. C:\Program Files\PTC\Creo 2.0\Common Files\M040).
`{proe_config(`*name*`)}`	The value of a Creo® configuration option, where *name* is the name of an option that can appear in a `config.pro` file.
`{proe_dir}`	The absolute path to the directory where Creo® is installed (e.g. C:\Program Files\PTC\Creo 2.0\Parametric).
`{proe_start_model_dir}`	If the Creo® option `start_model_dir` is defined, that value is used. If `{proe_common_dir}\templates` sub-directory exists, that value is used. If `{proe_dir}\templates` sub-directory exists, that value is used.
`{top_generic_cad_name}`	The name of the top family table generic if the Creo® model is a family table instance. If the model is not a family table instance, this is the same as `{cad_name}`.
`{user_id}`	The user's Teamcenter user ID.
`{temp_dir}`	The user's temporary directory, specified by the TEMP environment variable.

Template Map

The `template_map` tag specifies template models, also known as seed models or start models, that can be copied when a new model must be created. A template will usually be needed when the **Open** command attempts to fetch a model from Teamcenter, and the model's dataset is empty. When that happens, a template model of the required type will be copied and placed in the destination directory, with the copy being given the name that the missing model should have had.

A typical use case requiring configuration of the template map is top-down design, in which a product structure is created or extended in Teamcenter, to include item revisions with "empty" Creo® datasets (datasets without Creo® CAD files as named references). These empty datasets can be fetched and loaded into Creo® using the integration's **Update Assembly from BOM** feature, but the template map must be configured to tell the integration where to find the seed models.

Note

The *`template_map`* has no effect on Creo®'s **File→New** command. It is only used by commands in the **Teamcenter** menu.

A `template_map` contains zero or more `template` tags:

```
<template_map>
```

```
    <template cad_type="type" location/>
        .
        .
        .
</template_map>
```

Each `template` tag specifies a location where a template can be found, either on the local file system or in Teamcenter. Each `template` tag also has a `cad_type` attribute that specifies the type of Creo® model to be created from the template. The allowed values for type are the three-letter extensions that Creo® uses to identify its model files. For example, `cad_type="PRT"` means the template can be used to create part models.

If the template tag specifies a location in Teamcenter, more than one Creo® model type may be given for `cad_type`. Use a colon to separate each three-letter value from the next. For example, to indicate that an item in Teamcenter contains templates for parts, assemblies, and drawings, the attribute should be `cad_type="PRT:ASM:DRW"`.

The `cad_type` attribute should specify only one type when the location is on the local file system.

The models used as templates should not depend on other models. When a model specified in the `template_map` is copied to create a new model, that model alone is copied. No other models are copied with it.

If a family table generic is used in the `template_map`, the generic's Creo® file is copied in its entirety. The resulting new model will include the entire family table, and all instances will have the same names as in the original.

Note

If you decide to use a generic as a template model, consider removing all instances from the family table before adding it to the *`template_map`*.

A family table instance cannot be used as a template.

Using Templates from a Directory

A template model that is located in a directory is specified by the path to its file:

```
<template cad_type="type" path="path"/>
```

The file must exist, and *path* cannot contain any wildcards, but *path* may include keywords described in the Substitution Keywords section for general auxiliary files, if the keywords do not imply a specific model.

Note

Substitution keywords that must be resolved within a Creo® session, such as `{proe_config(name)}`, are not supported for use in the template_map, because in some scenarios, such as background export, a Creo® session will not be available.

If the file contains a Creo® dot version, the number extension must be included in template *path*. The following example is included in the installed ipem.xml. The templates specified are in the Creo® installation directory.

```
<template_map>
   <template cad_type="PRT"
      path="{proe_common_dir}/templates/mmns_part_solid_abs.prt"/>
   <template cad_type="ASM"
      path="{proe_common_dir}/templates/mmns_asm_design_abs.asm"/>
   <template cad_type="DRW" path="{proe_common_dir}/templates/a1_drawing.drw"/>
</template_map>
```

Note

Creo® 7.0 introduced new names for templates to support Absolute and Relative accuracy. The example in ipem.xml uses the templates for Absolute accuracy. Choose the template best suited for your work. The relative accuracy templates are the same as those in Creo® 6.0 and earlier releases.

Using Templates from Teamcenter

A template model located in Teamcenter is specified by its item ID, revision ID and model name:

```
<template cad_type="type"
  item_id="item" [item_revision_id="revision"] [model_name="name"]/>
```

where *name* is also the name of the dataset containing the file for the template model. The revision and model name are optional, and if not specified, all revisions and datasets in the item are searched for templates of the appropriate type. Wildcards may also be used in *item*, *revision,* and *name*, as if conducting a simple search in the Open dialog. If more than one revision of a model is found for a given type, the latest one is used. If several models with different names are found for a given type, one is arbitrarily selected. For example, suppose an item SEEDS is created in Teamcenter to hold the templates for parts and assemblies. This item can be referenced using a single `template` tag:

```
<template cad_type="PRT:ASM" item_id="SEEDS"/>
```

If SEEDS has a part dataset named MMNS_PART_SOLID and an assembly dataset named MMNS_ASM_DESIGN, the file contained in MMNS_PART_SOLID will be used to create parts and the file in MMNS_ASM_DESIGN will be used to create assemblies. If SEEDS has two revisions, the datasets from the later revision will be used. However, if SEEDS also has a part dataset named INLBS_PART_SOLID, either MMNS_PART_SOLID or INLBS_PART_SOLID will be arbitrarily selected as the template to use when creating a new part. In a situation such as this, it may be helpful to also specify the `model_name` attribute so only one of these models will be selected:

```
<template cad_type="PRT:ASM" item_id="SEEDS" model_name="MMNS*"/>
```

Dataset Map

Each type of Creo® CAD model maps to a unique dataset type and named reference in Teamcenter. The `type_map` and `dataset_map` tag define these one-to-one mappings. The `type_map` is not intended for customer modification and so it is not covered in this document.

A typical dataset map is shown below. None of these mappings is optional; however, the `allowed_workflows` tag within the dataset map *is* optional.

```
<dataset_map>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
  Release:Design Review" named_ref="AsmFile" pdm_type="ProAsm"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="PrtFile" pdm_type="ProPrt"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="DrwFile" pdm_type="ProDrw"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="FrmFile" pdm_type="ProFrm"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="SecFile" pdm_type="ProSec"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="LayFile" pdm_type="ProLay"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="DgmFile" pdm_type="ProDgm"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="MfgFile" pdm_type="ProMfg"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="MrkFile" pdm_type="ProMrk"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="RepFile" pdm_type="ProRep"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review" named_ref="GphFile" pdm_type="ProGph"/>
    <dataset allowed_item_types="Item" allowed_workflows="Quick
   Release:Design Review"named_ref="JTPART" pdm_type="DirectModel"/>
</dataset_map>
```

The mapping between CAD type and dataset type is established by the type map, which is not documented here. For each mapping between a Creo® CAD type and a Teamcenter dataset type, the allowable named reference, item types, and workflows are defined by an `entity` tag. The `pdm_type` identifies the dataset type and `named_ref` identifies the named reference for the Creo® model. Customers should change neither tag. The `allowed_item_types` and `allowed_workflows` tags are modifiable.

Allowed Item Types

The mandatory `allowed_item_types` attribute is a colon separated list that defines the list of item types to appear in the drop down on the save dialog. The first item type in the list is the default item type for the specified CAD type[5].

Note

Customers must configure the same default item type for CAD types which can be saved together to the same item ID, such as a drawing and its 3D model. Assigning different default item types to such CAD types can cause errors during save and import.

[5] Note that the ordering of the allowed_item_types list in the dataset map replaces the single iman.itemtypedefault ipem.properties preference that was used in earlier releases, providing more configuration flexibility.

Submit to Workflow

The integration can be configured to submit Item Revisions to a workflow process by adding “workflow” to the `table.columns.checkin` preference in ipem.properties.

The optional `allowed_workflows` tag is a colon separated list that defines the list of Teamcenter workflows to appear in the workflow column drop down on the save dialogs. The first workflow in the list is the default for the specified CAD type. Special characters such as ~!@#$%^&*()-=_+[]\{}|;’:”,./<>? in the workflow names must be escaped with a backslash \ in front of the character. A workflow with name “Testing:Workflow1” would be inserted into the `allowed_workflows` tag as follows:

```
allowed_workflows="Quick Release:Testing\:Workflow1:Design Review"
```

Note

Customers must configure the same default workflow for CAD types which can be saved together to the same item ID, such as a drawing and its 3D model. Assigning different default workflows to such CAD types can result in the same item revision being submitted to multiple workflows simultaneously, which is not the desired result.

The integration determines if an object is checked in, checked out, or released by evaluating the status of the dataset. Workflows that apply status to the objects should set the status on the Item Revision and the Creo® dataset. Failure to set the status on the dataset will enable the user to continue to work on the object. Teamcenter does not provide an out-of-the-box workflow that will simultaneously release both the Item Revision and the Dataset. See the Teamcenter documentation for more information on how to create and configure workflows.

Managing ipem.properties and ipem.xml in Teamcenter

The **ipem.properties** and **ipem.xml** files can be centrally managed in Teamcenter, instead of locally on each integration client. To manage settings in Teamcenter, attach **Creo Integration Settings** datasets to items named **Pro2_Properties** and **Pro2_XML**, then upload the configuration files as named references of the datasets. During login, the integration client will look for the items and datasets by name, and if found, will download and use them.

Either or both of the **ipem.properties** and **ipem.xml** files can be managed in this way, but neither is required. If a managed preference file is not found in Teamcenter, then the local file in the client installation directory will be used.

The following steps describe how to upload the **ipem.properties** file:

1. Create an Item with Item ID = **Pro2_Properties.** The initial Item Revision is automatically created by Teamcenter. The Item may be revised any number of times, as settings change, and the integration will always find and retrieve the file from the latest available revision.

2. Under the Item Revision, create a dataset of type **Creo Integration Settings** named **Pro2_Properties** (remove the /{RevisionID} extension from the dataset name).
3. Import the **ipem.properties** file as a named reference of the dataset.
4. Advise users to restart their sessions, to use the managed copy of **ipem.properties.**

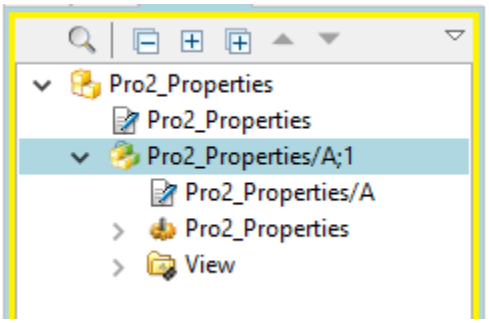


Figure 29 – Pro2_Properties Item, Item Revision, and Dataset

The following steps describe how to upload the **ipem.xml** file:

1. Create an Item with Item ID = **Pro2_XML**. The initial Item Revision is automatically created by Teamcenter. The Item may be revised any number of times, as settings change, and the integration will always find and retrieve the file from the latest available revision.
2. Under the Item Revision, create a dataset of type **Creo Integration Settings** named **Pro2_XML** (remove the /{RevisionID} extension from the dataset name).
3. Import the **ipem.xml** file as a named reference of the dataset.
4. Advise users to restart their sessions, to use the managed copy of **ipem.xml.**

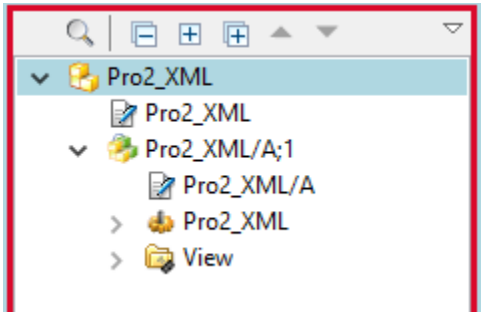


Figure 30 – Pro2_XML Item, Item Revision, and Dataset

When the integration settings are changed in Teamcenter, users are not notified until they restart their sessions and log into Teamcenter. A subset of the preferences in **ipem.properties** is used before logging into Teamcenter and cannot be changed during the login. These startup preferences are read from a copy of the **ipem.properties** file that was cached during the last login. If the integration detects that a startup preference has changed since the **ipem.properties** file was last cached, the user is alerted to exit Creo® and restart the integration so that the newly cached preferences are applied to the session.

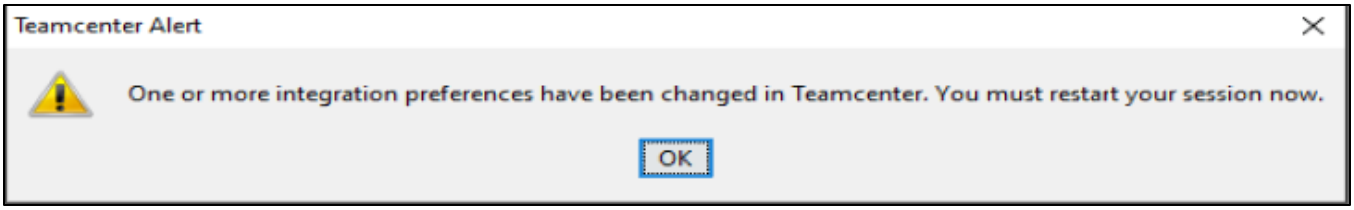


Figure 31 - Managed settings have been changed - the session must be restarted

A change to any of the following preferences requires users to restart their sessions:

- activeworkspace.enable
- activeworkspace.hosting.maxport
- activeworkspace.hosting.port
- iman.maxOperationBracketInactiveTime
- iman.maxOperationBracketTime
- iman.workspace.default
- iman.workspace.exception.folders
- log.append.date
- log.enable
- log.file
- pe.excludefrombom.enable
- pe.excludefrombom.familytable
- pe.udf.enable
- Pro2_CacheFolder
- registernew.enable
- windows.alwaysOnTop

If editing the files directly from Teamcenter, be sure to use an editor that will save the files as documented in each file's section of the **Administrator Guide**.

Note

Although the **Pro2_Properties** and **Pro2_XML** items, revisions and datasets will be found regardless of their Teamcenter owner, it is recommended to create and maintain these objects as the **infodba** user.

It is possible to create multiple copies of the **Creo Integration Settings** datasets under an item revision, each with a different set of preferences, and control access to each copy with Teamcenter Access Control tools such as ACLs. In this way, different settings can be implemented for different groups, roles, projects, etc. When this is done, it is essential that at all times, each user will find one

and only one copy of each file in Teamcenter. If multiple copies of **ipem.properties** or **ipem.xml** are available to a single user, behavior will be unpredictable.

Teamcenter Weight and Balance Support

Teamcenter 2406 and later introduced a Weight and Balance module. This enables roll up the properties for the assembly and view in RAC and Active Workspace.

Teamcenter Integration for Creo® supports the above feature by evaluating and writing mass properties of CAD models to Teamcenter on save. For the integration to access and save mass properties, the below preference must be set in ipem.properties

```
checkin.mass.properties.enable = true
```

Note

Mass properties cannot be collected for models that do not have materials set.

Teamcenter Integration for Creo® can only save mass properties for instances that are loaded into session. Instances open in session or referenced by assemblies open in session will get saved with mass properties. Instances added to the Save dialog though the Family Table->Add button will not have the mass properties saved.

Teamcenter Preferences

The integration installs Teamcenter site preferences to control the behavior of Teamcenter Server and Teamcenter Rich Client behavior.

Preference	Description
`Pro2_deferred_rename_enable`	The default value of `false` disables the **Creo®→Rename** command from the Teamcenter Rich Client. Setting the value to `true` allows users to rename Creo® datasets from the Teamcenter Rich Client. Before enabling please read Renaming Creo® Models in Teamcenter elsewhere in this document.
`PS_new_seqno_mode`	This Teamcenter preference controls how find numbers are generated when assemblies are saved. It can affect whether a Bill of Materials is packed or unpacked, in the Teamcenter Structure Manager application. See the Teamcenter documentation for valid values for this preference.
`appbar_title_contents`	If the `appbar_title_contents` contains the project the project will appear at the top of the integration dialogs when the user is working on an active project.
`ActiveWorkspaceHosting.Pro2.URL`	The integration's server install adds this preference and assigns a placeholder value of

	http://<CHANGE_ME_HOST>:<CHANGE_ME_PORT>/awc Customers who deploy Active Workspace for Creo® data management must replace this value with the appropriate values for their site.
ActiveWorkspaceHosting.URL	If a site uses multiple CAD integrations, this preference may be set instead of the ActiveWorkspacehosting.Pro2.URL. In that case the ActiveWorkspacehosting.Pro2.URL preference should be completely removed from the Teamcenter database. If it is found to exist by the integration, with its default placeholder value, login to Active Workspace will fail.
ETS.ASSEMBLYTYPES.SIEMENS.PROETOJT	This site preference is an array of assembly characteristics that the ets_request tag will create a translation request for. The possible values are **WireHarness**, **AssemblyFeatures**, **pmiOption**, **QuiltFlag** and **AttributeFlag** and all are included by default. Include only the values for the characteristics that justify translation within the site. The **AttributeFlag** results in all assemblies being translated. This preference only controls what is submitted for translation. The proetojtplmxml.config file must contain the equivalent values to ensure the resulting JT files include these characteristics. See the IPEMInstallationGuideForDispatcherClient for details.

PTC configuration settings

config.pro

Creo® configuration can influence integration behavior, in that it controls various aspects of file save and open, and updates to in-session models. Certain settings are required; other settings are recommended to improve usability and promote consistency, when working with the Teamcenter integration. Note that some of the indicated values are the PTC defaults, in which case they may be left unset.

Preference	Value	Req?	Description
modelcheck_enabled	yes	req	ModelCHECK is invoked during save to Teamcenter, to process non-referenced family table instances, and so the corresponding config.pro property must retain its default yes value.
allow_freeze_failed_assy_comp	yes	opt	CAD constraints that are lost due to partial loading or missing models do not result in

			packaged components within the Creo® parent assembly.
file_open_default_folder	working_directory	req	This setting is mandatory, to ensure that the native Creo® **File | Open** command defaults to the current workspace. Do not use a UNC path, as **config.pro** does not support UNC paths.
graphics	win32_gdi	opt	On Virtual Machines or machines with poor graphics cards, Creo® will crash when generating preview images. This setting prevents the crashes. PTC has indicated the setting was phased out with Creo® Parametric 2.0 M100, so PTC has hidden the option.
hide_pre_creo4_reps	no	req	This setting is required to open predefined representations such as Geometry Rep or Graphics Rep with the integration's **Open Rep** operation.
intf3d_in_import_as_facets	no	opt	JT files from other CAD integrations are imported into Creo® with BREP content, so that they may be used as reference geometry for native Creo® designs.
instance_search_exhaustive	yes	opt	Forces Creo® to perform an exhaustive search of family tables to find instances. This preference must be set to yes if users want to convert standalone parts or assemblies that exist in Teamcenter to family table instances.

nt_cache_dirs	N/A	req	This undocumented Creo® setting must not be used. Doing so can cause the integration's PTC API calls to fail.
override_store_back	yes	req	Forces all updates to be saved to the current working directory, which keeps auxiliary files (such as JT files) in the expected location.
retrieve_data_sharing_ref_parts	no	opt	Customers have observed save crashes of Creo 5 data in Creo 9 when this preference is set to ignore_missing
remember_replaced_components	no	opt	Unselects the **Remember These Components** option in the Creo® **Replace** dialog. When yes, the Creo® assembly will retain references to the replaced component. These references will persist as "dirty data" into going forward, therefore it is recommended to set this option to no.
regen_failure_handling	no_resolve_mode	opt	PTC is deprecating this preference. Starting in Creo® 6.0, regen_failure_handling is a hidden setting. Customers with Creo® 6 and later must no longer use this preference.
retrieve_instance_dependencies	instance_and_generic_deps	req	Required setting. When set to **instance_and_generic_deps**, both the instance and generic dependencies are retrieved when the instance is retrieved.
save_bitmap	alllevels	opt	Optional setting. The integration allows

			Creo® to create preview images for assemblies and parts. Setting this value improves performance of save operations. However, it will affect the performance of **Open** operations when attributes are mapped in the `pdmtocad` direction.
`save_bitmap_type`	`jpeg`	opt	When save_bitmap is set to “alllevels”, then the format required by Teamcenter is JPEG.
`save_drawing_picture_file`	`no`	req	Required setting. The default value of `embedded` is known to cause Creo® crashes when saving 3D models with “wrapped” text (like a decal), while also creating a PostScript and/or PDF file.
`save_file_iterations`	`yes`	req	The integration is designed to work with iterations. Failure to set this preference to `yes`, as it can cause Creo® to crash.
`save_object_in_current`	`yes`	req	Models that are opened from a read-only disk location are saved to the current working directory, during save to Teamcenter.
`save_objects`	`changed_and_ specified`	opt	A setting of “changed” is discouraged because it prevents users from converting family table instances to standalone parts. The default setting of “changed_and_specifie d” is best for the **Open** operation and for reliability of modification hints.
`try_noninst_on_fail`	`yes`	opt	Forces Creo® to look for standalone models if

			an instance is not found within its generic. This setting is required if users want to convert an instance to a standalone part or assembly.
windows_browser_type	*	opt	Hangs of the integration during Save can occur if set to chromium_browser. See release note in ReleaseNotes.pdf for link to PTC site.

config.sup

Creo® configuration variables can be locked by setting them in a config.sup file rather than a config.pro file. To ensure the integration behaves as designed, some configuration variables must remain unlocked. It is recommended not to use a config.sup at all.

Setting any of the following in config.sup will cause Creo® to crash.

- disable_search_path_check
- let_proe_rename_pdm_objects
- modelcheck_enabled
- override_store_back
- regen_failure_handling
- save_object_in_current

config_init.mc

Because the integration invokes ModelCHECK to read non-referenced family table instances, as well as (if configured) to validate models during save to Teamcenter, certain Creo® ModelCHECK options may need adjustment.

- ASM_BATCH_ALL N N N N
  - This setting prevents ModelCHECK from processing all components within an assembly, when only the assembly itself is being saved to Teamcenter.
- MC_ENABLE Y
  - The default setting of Y must be retained or save to Teamcenter will fail.
- MODE_RUN Y N N N
  - At a minimum interactive run mode must be enabled. The other modes may be either Y or N.

JT translation preferences

When JT files are translated during save to Teamcenter, the runtime tessellation configuration files are dynamically generated from two template files in the `{ipem_dir}`\pejt subdirectory on each client workstation:

- `proetojt.config`
  - Used for part-level JT translation
- `proetojtplmxml.config`
  - Used for assembly-level JT translation

Any configuration option which is not explicitly set in these files is inherited from the JT translator's hard-coded default settings[6]. In some cases, it may be necessary to modify the behavior by adding settings in these template files. Existing entries must not be deleted or modified. For example, a site may wish to enable PMI output from the JT translator. In that case, the following highlighted line may be added to the EAITranslator section of the proetojtplmxml.config file:

```
version "EAITranslator" 1.0.0 "EAITranslator"
EAITranslator {
    OutputDirectory = "{eai_plmxml_dir}"
    structureOption = "{eai_structure_option}"
    {eai_writeWhichFiles}
    includeBrep = true
    autoNameSanitize = false
    updateChangedPartsOnly = false
    singlePartsNoAssem = false
    pmiOption = "PART_AND_ASM"

}
```

To enable JT translator logging, uncomment the below lines in the *.config files within the pejt folder.

```
#    LogReporting = "ERROR,WARNING,INFO"
#    LogFileName = "PRO_PART"
```

Teamcenter DisplayName Constant

The **DisplayName** constant on each item type is read by the integration and displayed in the **Object** column in the integration dialogs. The constant defines the attributes on the item that are combined to uniquely identify items of the given type. Item types with multifield keys are not uniquely identified by the OOTB **DisplayName** constant. For item types with multifield keys, it may be useful to add the multifield key properties to the **DisplayName** constant, so users can distinguish between different items in the **Object** column of Teamcenter and integration dialogs.

Any attributes (other than multifield key attributes) configured in the **DisplayName** constant must be present in the `ipem_policy.xml` or `custom_policy.xml` file in the integration installation folder

[6] The tessPRO.config file in the JT translator's \etc subdirectory is completely ignored during integration-enabled JT translation.

(%IPEM_DIR%\policies) Missing properties must be added to the `custom_policy.xml` file. Failure to add such properties will result in assertion failures and degraded performance. In the `custom_policy.xml` file, the properties should be added under `ObjectType` where the name is the Item Type, as shown below.

```
<ObjectPropertyPolicy xmlns="http://teamcenter.com/Schemas/Soa/ObjectPropertyPolicy">
        <ObjectType name="ITI9_VendorItem">
                <Property name="Property1"/>
                <Property name="Property2"/>
                 :
                 :
                <Property name="Property"/>
        </ObjectType>
</ObjectPropertyPolicy>
```

Note

Errors or performance degradation could occur if the attributes referenced as pdm_name are not included in the policy file. Specifically, “assertion” failures may appear in the txdlog file if an attribute is missing from the policy file.

Multi-Site

Configuring Teamcenter Multi-Site to Export Creo® Models

If you are using Teamcenter multi-site data sharing to export or import items containing Creo® datasets, you must include the integration’s CAD-specific relations in the references to be traversed. The following types should be added to your **TC_relation_required_on_export** site preference:

```
IPEM_dependency
IPEM_mandatory_dependency
IPEM_master_dependency
IPEM_suppressed_dependency
Pro2_bulk
Pro2_diagram
Pro2_drawing_model
Pro2_external_srep
Pro2_format
Pro2_instance
Pro2_interchange_assembly
Pro2_interchange_membership
Pro2_layout
Pro2_markup
```

```
Pro2_membership
Pro2_merge
Pro2_mfg_reference
Pro2_skeleton
Pro2_substitute
Pro2_report
Pro2_reference
Pro2_udf
Pro2_udf_source
Pro2_udf_reference
Pro2_cem
Pro2_bom_exclusion
```

When exporting items using the **Tools** | **Export** | **Objects** command in the Teamcenter Rich Client's My **Teamcenter** application, or when importing items with the **Tools** | **Import** | **Remote** command, verify that the relations listed above appear in the **Advanced** tab of the **Export Preferences** dialog or **Import Remote Options** dialog, respectively.

Configuring Teamcenter Multi-Site for Transferring Ownership of Creo® Models

If users are allowed to transfer ownership of Creo® models from remote sites to your site, first configure multi-site to replicate items containing Creo® models, remembering when exporting and importing items to include the relation types described in the previous section.

At the site that owns the items you want to transfer, make sure your site is included in the list of sites defined by the **IDSM_permitted_transfer_sites** Teamcenter preference.

The site that owns the items must also grant the **TRANSFER_OUT** privilege for objects of the **POM_application_object** class. For example, in the Teamcenter Rich Client's **Access Manager**, this privilege might be granted by modifying the default Import/Export ACL as shown below.

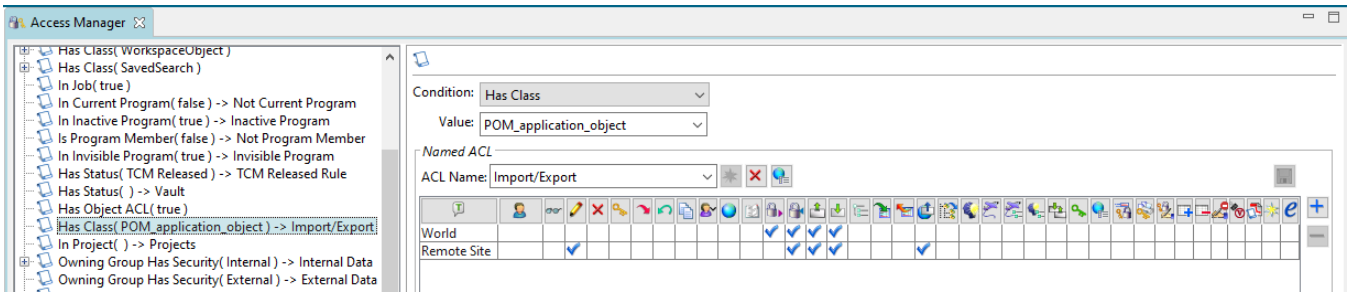


Figure 32 ACL Granting TRANSFER_OUT Privilege

Before starting Creo®, you may want to add a column to the Teamcenter Integration for Creo® dialogs so users can see the name of the site that owns each Creo® model. Do this by adding the site keyword to the list of columns defined by the `table.columns.std` user preference in the **ipem.properties** file. This user preference and others are described in the **ipem.properties** file, which can be found in the installation directory.

If you chose to add the **Transfer Ownership** command during installation, the command will appear in the shortcut menu when the user right-clicks on a model in the Teamcenter Integration for Creo® dialogs. If this command does not appear in the shortcut menu, run the installer again, choosing a Custom installation with only the **Configure Client for Multi-Site** option selected. Refer to the **User Guide's** section on **Using the Shortcut Menu** for more information on the **Transfer Ownership** command

Configuring replica icon in Integration dialogs

If users are opening models that are mastered on a replica site, enable the display of replica hints to aid them. In ipem.properties add **replica** to the **table.columns.xxxx** and **summary.table.rows.std**.

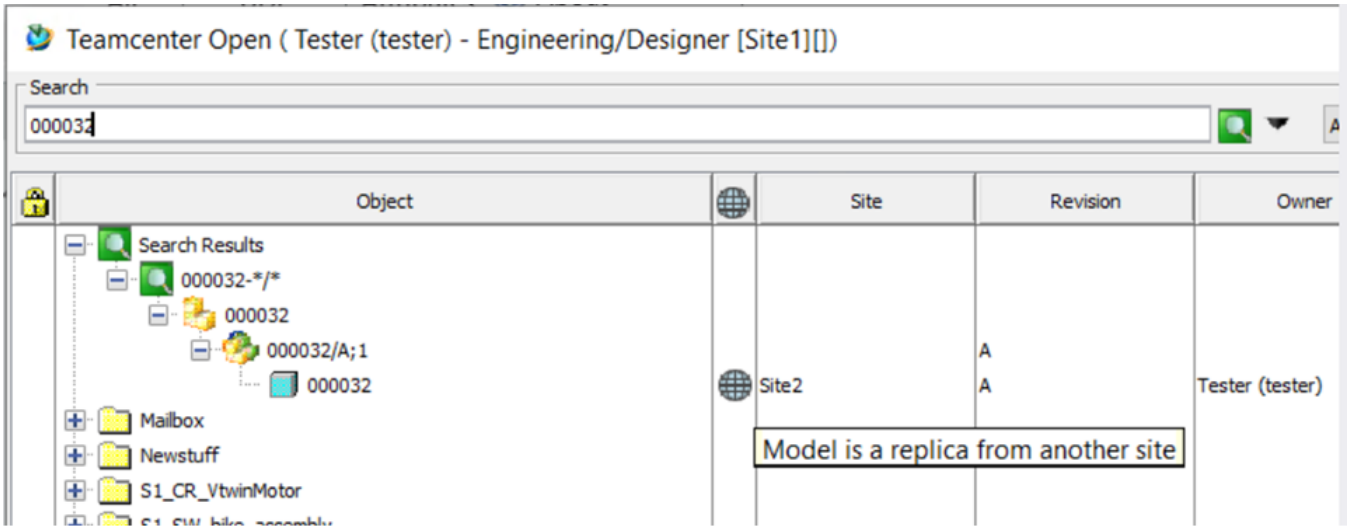


Figure 33 The replica hint as it appears in the open dialog in the table of objects. A text hint appears for the selected object in the summary panel.

Chapter 3

Chapter 4 Utility Functions

In addition to using the interactive integration in your Creo® session, you can also use several command line utilities to manage Creo® models in Teamcenter. These utilities are found in the same directory where the Teamcenter Integration for Creo® is installed. These are:

- The `ipemexport` utility for exporting Creo® models from Teamcenter.
- The `ipemimport` utility for saving Creo® models to Teamcenter.
- The `ipemrename` utility for renaming Creo® models in Teamcenter.

If the integration's installation directory is in your path, you can run any of these utilities by opening a shell or command window for your operating system and typing the utility's name with appropriate arguments on the command line. Arguments are shown in detail in the following sections under Usage. Arguments enclosed in square brackets [] are optional, a vertical bar | separates alternate values that can be used for an argument, and text shown in italics is to be replaced by an appropriate value when you use the command. For example,

ipemexport –u=*username* **–p**=*password* [**-export** true|false]

shows the `ipemexport` utility's `-export` argument is optional because it is enclosed in brackets, and that its value can be one of the literal words `true` or `false`, while *username* and *password* must be replaced by your Teamcenter user name and password.

Exporting Creo® Models from Teamcenter with ipemexport

The `ipemexport` utility exports Creo® models from Teamcenter. Datasets may also be checked out (reserved) when exporting models. Models that are checked out by `ipemexport` may be checked back into Teamcenter with the `ipemimport` utility.

Usage

```
ipemexport [-login] [-u=username -p=password [-g=groupname]]
 [-role=rolename]
 [-project=project_id]
 [-help]
 [-delete_files] [-skip_files] [-export_dir=path]
```

[-export=true|false] **[-export_related**=true|false]

[-checkout=true|false] **[-checkout_related**=true|false]

[-change_id=string] **[-comment**=string] **[-config**=rule]

[-no_filters] [-filter=name] **[-levels**=limit]

[-item_id=id] **[-item_revision_id**=id]

[-model_name=name] **[-model_type**=datasetType]

[-dry_run] [-output=instructionfile] **[-log**=logfile] **[-summary]**

[-incomplete_operation_fatal | -incomplete_operation_ok]

[-servername=servername]

[-arrangement=arrangement]

file1 file2 ...

Description

The `ipemexport` utility will export a Creo® model, and the models on which it depends, from Teamcenter to a local directory. The model to export may be identified with the `-item_id`, `-item_revision_id`, `-model_name` and `-model_type` options on the command line, or several models may be specified in *instruction files*, given by arguments *file1*, *file2*, etc., following any command line options. Instruction files are text files that list models to export, one model per line.

Some options affect only *explicitly specified* models identified on the command line or in instruction files. Other options affect *implicitly specified* models that are not identified on the command line or in instruction files, but which are also exported because explicitly specified models depend on them.

The default action is to export all explicitly specified models and the models on which they depend, but this can be changed using the `-export` option for explicitly specified models and the `-export_related` option for implicitly specified models. To reserve models, the `-checkout` option reserves explicitly specified models and `-checkout_related` reserves implicitly specified models. Models that have been reserved may be checked back into Teamcenter using the ipemimport utility.

The `ipemexport` utility uses the same mapping file and user preferences used by the Teamcenter Integration for Creo®.

One significant difference between the `ipemexport` utility and the Teamcenter Integration for Creo® is that `ipemexport` does not copy attribute values from Teamcenter to parameters in the Creo® models.

The `ipemexport` options are described below.

-arrangement=*name*
: The name of the arrangement in Teamcenter that corresponds to a Creo® simplified representation within the assembly being exported. Only models included within the arrangement are exported. The user must ensure that the correct simplified representation is selected when opening the assembly from the local directory; otherwise, Creo® may report missing models (optional).

-change_id=*id*
: Specifies a change ID to assign to all datasets that are checked out (optional).

-checkout=true | false

True specifies that explicitly specified models should be reserved (optional). The default is false, which means that these models will not be reserved. Note that if `-export` is false, this option can be used to reserve a dataset without exporting the model file in it. This option does not affect implicitly specified models. To reserve implicitly specified models, use the `-checkout_related` option.

-checkout_related=true | false

True specifies that implicitly specified models should be reserved (optional). The default is false, which means that these models will not be reserved. Note that if `-export_related` is false, this option can be used to reserve datasets without exporting the model files in them.

-comment=*string*

Specifies a comment to assign to all datasets that are checked out (optional).

-config=*rule*

Specifies the name of a configuration rule (optional). The default is to use the rule specified by the `iman.bomrevisionruledefault` user preference or the default Teamcenter rule. If the revision rule contains spaces, it must be enclosed within double quotes.

-delete_files

Specifies that conflicting files already in the export directory should be deleted before any models are exported (optional). Files that do not conflict with the models to be exported are not affected. If neither `-delete_files` nor `-skip_files` is specified, the program will abort if any conflicts are found. If both `-delete_files` and `-skip_files` are given on the command line, the last one specified takes effect.

-dry_run The utility goes through most of the actions it would ordinarily execute, but it does not export any files or reserve datasets (optional). When used in conjunction with the `-output` option, an instruction file can be generated for use in a subsequent run of this utility.

-export=true | false

True specifies that explicitly specified models should be exported, which is the default (optional). False means that these models will not be exported. Note that `-checkout` may be true even if `-export` is false. This option does not affect implicitly specified models. To export implicitly specified models, use the `-export_related` option.

-export_dir=*path*

Specifies the path to the export directory (optional). If not specified, files will be exported to the current directory.

-export_related=true | false

True specifies that implicitly specified models should be exported, which is the default (optional). False means that these models will not be exported. Note that the `-checkout_related` option may be true even when `-export_related` is false.

-filter=*name*

Specifies a filter to use when collecting implicitly specified models (optional).

Each use of this option adds another filter to any filters previously specified. Later filters can override the actions of previous filters. The default filters are those specified by the `filters.checkout.default` user preference but note that the `-no_filters` option can erase those prior filters. See the `filters.checkout.choices` user preference for the names of filters that can be used with this option. User preferences are described in the `ipem.properties` file located in the directory where the Teamcenter Integration for Creo® is installed.

-g=*groupname*

Specifies the group name to use when connecting to Teamcenter (optional). Ignored when `-login` is specified.

-help Prints usage information and exits.

-incomplete_operation_fatal

Causes the program to exit when an operation on a model cannot be completed due to an error (optional). The default is `-incomplete_operation_ok`.

-incomplete_operation_ok

Causes the program to continue when an operation on a model cannot be completed due to an error (optional). This is the default.

-item_id=*id*

Specifies the Teamcenter item ID of a model to export (optional). This option may be used only once on the command line, but wildcards are allowed. Double quotes in the `item_id` string must be escaped with a preceding backslash such as `\"`.

-item_revision_id=*rev*

Specifies the Teamcenter revision of a model to export (optional). This option may be used only once on the command line, but wildcards are allowed.

-levels=*limit*

Specifies the number of levels to include in the hierarchy of dependencies when collecting implicitly specified models (optional). If not specified, or if *limit* is a negative number, the number of implicitly specified models is unlimited. A positive number sets the maximum number of levels of implicitly specified models that can be collected. This option has the same purpose as the `limit.levels` user preference, described in the `ipem.properties` file located in the directory where the Teamcenter Integration for Creo® is installed, although ipemexport ignores `limit.levels`. The limit can only be set using this command line argument.

-log=*logfile* Defines the name of the log file that will be created (optional). If not specified, the log file will be `ipemexport.txt` in the current directory. Logging cannot be disabled, and warning messages cannot be suppressed, but otherwise the logging preferences in `ipem.properties` have their usual effect.

-login Uses the Teamcenter login dialog. When this option is specified, the `-u`, `-p`, `-g`, and `-role` options are ignored. Use `-login` to avoid having to escape special characters that may be included in username, password, group, or role. Some special characters cannot be escaped and require the use of `-login`.

-model_name=*datasetname*

Specifies the name of a Creo® model. This is the name alone, without any extension or path. For example, the name of a model stored in the file `widget.prt.2` is `WIDGET`. The model name is the same as the dataset name.

-model_type=*type*

Specifies the dataset type or the three-letter extension that identifies the Creo® model type. For example, the type of a model stored in the file `widget.prt.2` is `PRT`, and its dataset is of type ProPrt, so *type* for a Creo® part model may be either `PRT` or `ProPrt`.

-no_filters

Causes any previous list of filters to be erased, such that no filters will be used when collecting implicitly specified models. However, a new list of filters will be created if this option is followed by one or more `-filter` options.

-output=*instructionfile*

Defines the name of an instruction file to be written (optional). This file lists each model exported or reserved from Teamcenter.

-p=*password* Specifies the password to use when connecting to Teamcenter. Ignored when `-login` is specified. Password may contain special characters as long as the password is enclosed in double quotes. Double quotes cannot be contained in the password string.

-project=*project_id*

Specifies the project to use when connecting to Teamcenter (optional).

-role=*role_name*

Specifies the role name to use when connecting to Teamcenter (optional). Ignored when `-login` is specified.

-servername=*name*

The Teamcenter server name. If not specified, the default server is that of the first server in the `client_specific.properties` file, found in the Teamcenter Portal installation directory. The name should be one of specified by a property such as `DEFAULT_SERVER.NAME` or `HTTP_SERVER_1.NAME` in the `client_specific.properties`. Ignored when `-login` is specified.

-skip_files

Specifies that conflicting files already in the export directory should be preserved by skipping export of the corresponding files from Teamcenter (optional). Files that do not conflict with the models to be exported are not affected. The default is to not skip files. If neither `-delete_files` nor `-skip_files` is specified, the program will abort if any conflicts are found. If both `-delete_files` and `-skip_files` are given on the command line, the last one specified takes effect.

-summary

Specifies that a summary dialog be displayed upon completion.

-u=*username* Specifies the user name to use when connecting to Teamcenter. Ignored when `-login` is specified.

Family Tables

Every Creo® model in Teamcenter is represented by a dataset, even models that are family table instances. However, in the case of a family table instance, the dataset does not contain the model's file. Instead, the instances in a family table depend on the family table generic, and it is in the dataset for the generic that the Creo® model file can be found.

If you "export" a family table instance, ipemexport will actually export the model file from the family table generic's dataset. If you reserve a family table instance, ipemexport will also reserve the generic.

Multiple Model Versions

Only one version of any given Creo® model will be exported in one run of the `ipemexport` utility. For example, if several revisions of a model are specified on the command line or in instruction files, only the first one encountered will actually be exported.

Instruction Files

An instruction file is a UTF-8 encoded text file that lists Creo® models. Each line in the file should identify one model. Blank lines and comments are also allowed. The comment character is #. Any text between the comment character and the end of the line will be ignored. A model is identified by several keyword=value pairs. A value containing spaces must be enclosed in double quotes, and keywords are one of the following:

checkout	This is the same as the –checkout command line option, but it applies only to the model specified on the line where this keyword appears. The value may be either true or false. If this keyword is not specified, the value of the –checkout command line option applies.
export	This is the same as the –export command line option, but it applies only to the model specified on the line where this keyword appears. The value may be either true or false. If this keyword is not specified, the value of the –export command line option applies.
item_id	The value is the item ID.
item_revision_id	The value is the revision.
model_name	The value is the name of the Creo® model. This is the name alone, without any extension or path. For example, the name of a model stored in the file widget.prt.2 is WIDGET. The model name is the same as the dataset name.

model_type	The value is the dataset type or the three-letter file extension that identifies the Creo® model type. For example, the type of a model stored in the file widget.prt.2 is PRT, and its dataset is of type ProPrt, so the value may be either PRT or ProPrt.
property	The property keyword identifies a multifield key property and value. The syntax is `property[class.name] = value`. The property is used with other key properties and keywords to unique identify a model, when multiple items share the same item ID.

If a value contains the space, #, or = characters, then the value must be enclosed in double quotes. Double quote characters within the value must use the \ character to escape the double quote.

A simple instruction file is shown below:

```
# Sample ipemexport instruction file
item_id=702283 item_revision_id=B model_name=BOLT_A model_type=PRT
item_id=702284 item_revision_id=A export=false checkout=false
item_id=702285 item_revision_id=A
item_id=702286 item_revision_id=C model_name=BOLT_A model_type=PRT
property[item.iti9_Vendor]= "Vendor1"
```

The first line is a comment. The second line identifies a particular model, the Creo® part BOLT_A in revision B of item 702283. No `checkout` or `export` keyword is given on this line, so the command line options `-checkout` and `-export` will be used to decide whether to export and/or reserve this model. By default, `ipemexport` will export the model without reserving it.

The third line specifies an item ID and revision, but the model name and type are not given. If revision A of item 702284 has several Creo® datasets, this line affects all of them. For example, if revision A contains both a part and a drawing, neither will be exported or checked out because `export` and `checkout` are both false on this line.

The fourth line also identifies a model using only its item ID and revision. Since no `export` or `checkout` keyword is included on this line, the model will be exported or reserved according to the command line options `-export` and `-checkout`.

The fifth line identifies a model using its item ID, revision and the multifield key property item.iti9_Vendor.

Note

When a model's item cannot be unique identified by its item ID alone, the `property` keyword may be used to specify additional key properties, as defined by the Multifield Key specification of the item type. However in this case, the `model_name` and `model_type` keywords must also be included in the instruction file, or the utility will fail to distinguish between multiple items with the same ID and will export all of them.

Examples

The following example shows a simple use of this export utility.

```
ipemexport –u=infodba –p=infodba –item_id=702283 –item_revision_id=B
```

This command will export the Creo® model(s) in revision B of item 702283, plus all models on which it depends. The files will be placed in the current working directory. None of the models will be reserved. If the models in 702283 should be reserved, the command becomes:

```
ipemexport –u=infodba –p=infodba –item_id=702283 –item_revision_id=B \
      -checkout=true –comment="Changes per 4283"
```

This reserves only the model in revision B of 702283, but the models on which it depends are still exported. If the Creo® model in revision B should be reserved without being exported, and the models on which it depends also should not be exported, the command would be:

```
ipemexport –u=infodba –p=infodba –item_id=702283 –item_revision_id=B \
      -checkout=true –export=false –export_related=false \
      –comment="Changes per 4283"
```

If 702283 is a family table instance, the command above will also reserve the family table generic regardless of how `–export_related` is set.

Suppose an instruction file named `exports.txt` specifies the models to export. In this case, the command would be:

```
ipemexport –u=infodba –p=infodba exports.txt
```

To find out what will really be exported or reserved prior to executing the command above, use the `–dry_run` option and ask ipemexport to generate an instruction file:

```
ipemexport –u=infodba –p=infodba –dry_run –output=out.txt exports.txt
```

Importing Creo® Models into Teamcenter with ipemimport

The `ipemimport` utility imports Creo® models into Teamcenter. This non-interactive utility saves models to Teamcenter after opening them in Creo®. During the import process, items, item revisions, and datasets may be created, parameter values may be copied to attributes in Teamcenter, and BOMs may be updated.

Note

The ipemimport utility is not suitable for data migration. It is designed to support the import of relatively small, self-contained sets of Creo® data, for example, importing a subassembly from a supplier. Large-scale data migration from a legacy PDM system, or from disk, requires extensive preparation, cleanup, and testing, and is beyond the scope of the ipemimport utility.

Usage

ipemimport [**-login**] [**-u**=*username* **-p**=*password* [**-g**=*group*]] [**-role**=*role_name*]
[**-project**=*project_id*]
[**-help**]
[**-dry_run**] [**-output**=*instructionfile*] [**-log**=*logfile*]
[**-folder**=*folder*] [**-overwrite** | **-revise**]
[**-precise_bom**=true|false]
[**-no_create**] [**-no_boms**]
[**-auxiliary_files_only**]
[**-no_family_table_expansion**]
[**-new_itemid**]
[**-existing_dataset_fatal** | **-existing_dataset_ok**]
[**-changed_dataset_fatal** | **-changed_dataset_ok**]
[**-incomplete_operation_fatal** | **-incomplete_operation_ok**]
[**-missing_child_fatal** | **-missing_child_ok**]
[**-missing_file_fatal** | **-missing_file_ok**]
[**-unmapped_type_fatal** | **-unmapped_type_ok**]
[**-servername**=*servername*]
[**-except**=*exceptfile*] *file1 file2 ...*

Description

The arguments *file1*, *file2*, ..., are Creo® models to import, directories containing Creo® models, or *instruction files*. An instruction file is a text file in which each line is either the path to a Creo® model

file or a directory containing Creo® models. This utility opens all of the models specified by these arguments in a single, non-graphical Creo® session, obtains information concerning parent-child relationships, parameter values, family tables, and so on, then imports the models into Teamcenter.

Models are imported in three steps. In the first step, Teamcenter objects (items, item revisions, and datasets) are created as needed and the model files are imported into the datasets. In the second step, parameter values are copied from Creo® to attributes of the Teamcenter objects. The final step updates BOMs and other references between objects. Two of these steps can be disabled with the `-no_create` or `-no_boms` options.

When importing a model into an existing dataset, you must either use the `-overwrite` option or check out the dataset prior to running `ipemimport` (the `ipemexport` utility may be used to check out datasets). Models will not be imported into datasets that are reserved by another user. In addition, if you reserve a dataset using check out with export, the model can only be imported from the export directory that was specified at the time you reserved the dataset, and the model will be deleted from that directory after it is imported.

The other arguments for `ipemimport` are described below.

`-auxiliary_files_only`
: Only auxiliary files are imported, without Creo® models. This also implicitly sets `-no_boms`.

`-changed_dataset_fatal`
: Causes the program to exit when a model's dataset already exists in Teamcenter and appears to have changed since the model was exported (optional). The default is `-changed_dataset_ok` when the `checkin.pdmversion.ignore` user preference is `always` and `-changed_dataset_fatal` when the `checkin.pdmversion.ignore` user preference is `never` or `prompt`.

`-changed_dataset_ok`
: Causes the program to continue regardless of whether a model's dataset appears to have changed since the model was exported (optional). This is the default when the `checkin.pdmversion.ignore` user preference is `always`.

`-dry_run`
: The utility goes through most of the actions it would ordinarily execute, but it does not import any files or other data to Teamcenter (optional). When used in conjunction with the `-output` option, an instruction file can be generated for use in a subsequent run of this utility.

`-except`=*exceptfile*
: Specifies Creo® models to skip. The *exceptfile* argument may be a Creo® model file, an instruction file, or a directory (optional). The models specified by this option will be opened in Creo® but will not be imported to Teamcenter. This option may be repeated to add more models to the exception set.

`-existing_dataset_fatal`
: Causes the program to exit when a model cannot be imported because its dataset already exists (optional). This argument is ignored when `-no_create` or `-overwrite` is specified. The default is `-existing_dataset_ok`.

`-existing_dataset_ok`
: Causes the program to continue when a model cannot be imported because its dataset already exists (optional). This is the default.

-folder=*folder*

Specifies the name of a Teamcenter folder, belonging to the same user and group as the session user, into which new items will be placed. This parameter is optional, and must be the name of the folder only, not a path to a nested folder. If the session user and group own more than one folder of this name, then the first folder found will be used.

-g=*groupname*

Specifies the group name to use when connecting to Teamcenter (optional). Ignored when `-login` is specified.

-help Prints usage information and exits.

-incomplete_operation_fatal

Causes the program to exit when an operation on a model cannot be completed due to an error (optional). The program will not exit if there are any failures during object creation. However, it will exit if there are failures during the validation stage. The default is `-incomplete_operation_ok`.

-incomplete_operation_ok

Causes the program to continue when an operation on a model cannot be completed due to an error (optional). This is the default.

-log=*logfile* Defines the name of the log file to be written (optional). If not specified, the log file will be `ipemimport.txt` in the current directory. Logging cannot be disabled, and warning messages cannot be suppressed, but otherwise the logging preferences in `ipem.properties` have their usual effect. The `client_socket log` is always created in the current directory.

-login Uses the Teamcenter login dialog. When this option is specified, the `-u`, `-p`, `-g`, and `-role` options are ignored. Use `-login` to avoid having to escape special characters that may be included in username, password, group, or role. Some special characters cannot be escaped and require the use of `-login`.

-servername=*name*

The Teamcenter server name. If not specified, the default server is that of the first server in the `client_specific.properties` file, found in the Teamcenter Portal installation directory. The name should be one of specified by a property such as `DEFAULT_SERVER.NAME` or `HTTP_SERVER_1.NAME` in the `client_specific.properties`. Ignored when `-login` is specified.

-missing_child_fatal

Causes the program to exit when a model being imported depends on another model that cannot be found (optional). The default is `-missing_child_ok` when the `checkin.ignoremissing` user preference is `always` or `prompt` and `-missing_child_fatal` when the `checkin.ignoremissing` user preference is `never`.

-missing_child_ok

Causes the program to continue when a model being imported depends on another model that cannot be found (optional). This is the default when the `checkin.ignoremissing` user preference is `always` or `prompts`.

-missing_file_fatal

Causes the program to exit when a file specified on the command line or in an instruction file cannot be found (optional). This is the default.

-missing_file_ok

Causes the program to continue when a file specified on the command line or in an instruction file cannot be found (optional). The default is `-missing_file_fatal`.

-new_itemid

Creates new items, with item IDs determined by naming rules defined in Teamcenter (optional). The default is to create new items with IDs matching the CAD file names, but this behavior can be overridden by specifying `-new_itemid`, or by using an instruction file. The advantage of using `-new_itemid` is that item IDs need not be predefined, as is the case with an instruction file. Teamcenter will handle generating the new values, according to the item type assigned to the incoming data. If the item type has naming rules, the naming rule will be defaulted by Teamcenter to be the last pattern used. CAD files will be renamed to match the assigned item IDs by default, but this behavior may be overridden by setting `iman.autorename.model=never` in `ipem.properties`.

-no_boms Skips the step in which BOMs and other references are updated in Teamcenter (optional).

-no_create

Skips the step in which items, item revisions, and datasets are created in Teamcenter (optional). This also skips the step in which Creo® model files are imported into the datasets. This option is incompatible with `-revise`.

-no_family_table_expansion

Skips the step in which all family table instances are regenerated (optional). The default is to regenerate all family table instances in the Creo® session, then to save every family table instance to Teamcenter as a separate dataset. This option is ignored if `checkin.familytable.requireall=always` in `ipem.properties`. This option is recommended for importing only referenced (used) instances.

-output=*instructionfile*

The absolute path to an instruction file to be written (optional). This option is to be used in conjunction with `-dry_run` to generate an instruction file template of the models that would be imported into Teamcenter. If used without the `-dry_run` option, the instruction file cannot be used to determine whether a given model was successfully imported.

-overwrite

Causes existing model files already in Teamcenter to be replaced if a new file is imported (optional). A checkout of the object occurs during the overwrite. The default is to skip models that already exist in the latest revision unless they are checked out to the user. Models that are checked out do not require `-overwrite` to be imported.

-p=*password* Specifies the password to use when connecting to Teamcenter. Ignored when `-login` is specified. Password may contain special characters as long as the

password is enclosed in double quotes. Double quotes cannot be contained in the password string.

-precise_bom=true | false
True causes precise BOMs to be created (optional). The default is true.

-project=*project_id*

Specifies the project to use when connecting to Teamcenter (optional).

-revise Causes a new revision to be created for each model imported into Teamcenter (optional). The default is to import into the latest revision, skipping any model if the latest revision already contains a version of the model. This option is incompatible with `-no_create`.

-role=*role_name*

Specifies the role name to use when connecting to Teamcenter (optional). Ignored when –login is specified.

-u=*username* Specifies the user name to use when connecting to Teamcenter. Ignored when `-login` is specified.

-unmapped_type_fatal
Causes the program to exit when a model cannot be imported because the model's type has not been mapped to a Teamcenter dataset type (optional). This is the default.

-unmapped_type_ok
Causes the program to continue when a model cannot be imported because the model's type has not been mapped to a Teamcenter dataset type (optional). The default is `-unmapped_type_fatal`.

Item IDs, Names, Types, and Revisions

A model's item ID, name, type, and revision can be specified in an instruction file or by mapping parameters from the Creo® model to these attributes in Teamcenter. See the section on **Instruction Files** below for more information on how to do this in an instruction file, or refer to the section on **Attribute Mapping** for more information on how to set up an attribute map. If the model's item ID, name, type, or revision is not specified, the following defaults apply.

- The default item ID is assumed to be the same as the model name. The model name is also used as the default item name. For example, if a part file `widget.prt` is imported into Teamcenter, the item ID and item name will both be WIDGET.
- The default item type is determined by a combination of the model's CAD type and the first item type to appear in the `allowed_item_types` list in the dataset map within the ipem.xml file. See the section on the map file for more details.
- The default revision is the latest revision of an item unless the `-revise` option is specified.

When several Creo® models have the same item ID, they will be imported into the same item. For example, if `widget.drw` contains a drawing of `widget.prt` and no other item ID is specified for either model, both will be stored in the item whose ID is WIDGET.

Attributes

This utility will copy parameter values from Creo® models to properties in Teamcenter. The attribute values that are copied are specified by the Teamcenter Integration for Creo® mapping file. The default mapping file is `ipem.xml` in the Teamcenter Integration for Creo® installation directory.

If a model was skipped during the file import step because its item revision already has a dataset for the model, it will also be skipped during the step that copies attribute values. Specifying the `-overwrite`, `-revise`, or `-no_create` option will ensure no models are skipped, except for those specified with the `-except` option.

BOMs

BOMs will be created or updated by this utility. The types of models that are added to the BOMs, and the BOMs that are affected, are specified by the Teamcenter Integration for Creo® mapping file. The default mapping file is `ipem.xml` in the Teamcenter Integration for Creo® installation directory.

BOMs are precise by default. This can be changed by using the `-precise_bom` option or by setting the `iman.precisebom` preference, described in the `ipem.properties` file in the Teamcenter Integration for Creo® installation directory.

If a model was skipped during the file import step because its item revision already has a dataset for the model, it will also be skipped during the step that updates BOMs. Specifying the `-overwrite`, `-revise`, or `-no_create` option will ensure no models are skipped, except for those specified with the `-except` option.

Family Tables

If a model contains a family table, all of the instances are also imported into Teamcenter, except for those instances specified with the `-except` option in instruction files. Each family table instance is treated as a model in its own right. An item, item revision, and dataset will be created for it, although the Creo® model file will be stored only in the dataset of the family table's top generic.

This behavior can be changed with the `-no_family_table_expansion` option. When specified, `ipemimport` does not attempt to import all family table instances, but instead it imports only those instances that are implicitly opened in the Creo® session because they are used by other models. When `-no_family_table_expansion` option is used with an instruction file, all referenced instances and all instances listed in the instruction file are imported.

UDF Files

Creo® UDF (.gph) files are indirectly opened by ipemimport. The Creo® APIs cannot directly open UDF files. UDF files get imported when the referencing model is opened and imported. If UDF information is provided in an instruction file, it will get used when the UDF is imported with the referencing model. Not all UDF files can be imported. The Creo® APIs do not provide a means to determine the name of the UDF file. Instead, the integration tries to match the UDF by filename. The referencing model must start with the same name as the GPH file. For example, the integration will find udf1.gph.1 if the referencing model is udf1_gp.prt.1. It is the "udf1" that is matched.

A warning is provided when the integration skips the direct open of a UDF. After import is complete validation should be performed to confirm that the UDF files got imported. Any UDF files not imported can only be saved into Teamcenter by performing an interactive save of the referencing model.

Multiple Model Versions

Version numbers are ignored in Creo® file names. This program loads only the highest-numbered version of a Creo® model when several versions occur in a directory. For example, if a directory contains `widget.prt.5` and `widget.prt.6`, only `widget.prt.6` will be imported into Teamcenter, even if `widget.prt.5` is specified on the command line.

Models are loaded in the order in which they appear on the command line, which means that if two directories contain different versions of a model, the model in the directory specified first will be loaded first, preventing the other directory's version from being loaded. For example, if directory A contains `widget.prt.1` and directory B contains `widget.prt.2`, `widget.prt.1` will be loaded instead of `widget.prt.2` if directory A appears before directory B on the command line.

Auxiliary Files

The `ipemimport` utility will create and import auxiliary files when importing models if an auxiliary file map exists. For more information on auxiliary files, refer to the section on the Map File.

If you do not want to import models, but only auxiliary files, consider using the `import_file` utility, which is included with Teamcenter. Note that the Teamcenter Integration for Creo® can be configured, via the auxiliary file map, to provide the item ID, revision, dataset name, dataset type, and named reference for each auxiliary file required when a model is saved to Teamcenter. This information can be used later to import auxiliary files with `import_file`.

Instruction Files

An instruction file is a UTF-8 encoded text file that lists Creo® model files or directories containing Creo® models. Each line in the file should contain the path to one file or directory. Blank lines and comments are also allowed. The comment character is #. Any text between the comment character and the end of the line will be ignored.

If the path to a file or directory contains spaces, it must be enclosed in double quotes. For example, the following path does not have to be quoted:

```
D:\Models\widget.prt.1
```

but this path must be quoted:

```
"D:\Pro  Models\widget.prt.1"
```

The path to a Creo® model may be followed by several optional arguments on the same line. Each argument has the form *keyword=value*, where *keyword* is one of the following:

dataset_desc	The value is the dataset description. This is ignored if the dataset already exists.
item_desc	The value is the item description. This is ignored if the item already exists.
item_name	The value is the item name. This is ignored if the item already exists.

item_id	The value is the item ID. If an item type has associated naming rules, the `item_id` must be compliant to one of the naming patterns defined by the naming rules.
item_revision_id	The value is the revision.
item_type	The value is the item type. For best results, `item_type` should be specified.
model_name	The value is the name of the Creo® model. This is the name alone, without any extension or path. For example, the name of a model stored in the file widget.prt.2 is WIDGET. This option is redundant if the file name is given. It cannot be used to rename the model.
model_type	The value is the three-letter extension that identifies the Creo® model type. For example, the type of a model stored in the file widget.prt.2 is PRT. This option is redundant if the file name is given. It cannot be used to change the type of the model.
property[class[:type].name]	The property keyword is used to specify Teamcenter properties other than the keyword properties listed above. The *class[:type].name* string specifies the Teamcenter property. Refer to the pdm_name section under Attribute Mapping for details on *class[:type].name* syntax. If the property is an array, the value must be included in curly braces {} and comma separated. If required properties or multifield keys are attached to the target item type, via Teamcenter configuration, then the property keyword may be required in order to import CAD models with this utility. If the property is a multifield key, then the property is used with Item ID or model name to import a model to a unique Item. The syntax is `property[class[:type].name] = value`. If a value contains spaces, it must be enclosed in double quotes.
units_length	The value is the units used to measure lengths in the Creo® model. If not specified, the default is taken from the pe.units.length user preference, which is described in the ipem.properties file in the Teamcenter Integration for Creo® installation directory. The value may be one of the following strings: cm foot inch m micron mil mm Alternatively, the value may be a real number, which specifies the number of inches per unit of length used in the model.

If a value contains the space, #, or = characters, then the value must be enclosed in double quotes. Double quote characters within the value must use the \ character to escape the double quote.

The Creo® file name does not have to be given if the model_name and model_type options are included on the line. Although these two options are not sufficient to identify the file to load, they can be used to refer to a model that is loaded implicitly or explicitly by a file specified elsewhere in the arguments given to ipemimport. For example, suppose three different revisions of widget.prt are stored in directories named reva, revb, and revc, and a single instruction file named widget.txt has been created to specify the item ID to use for the model:

```
model_name=WIDGET model_type=PRT item_id=184786
```

The ipemimport utility can then be executed three times to load the revisions:

```
ipemimport –u=infodba –p=infodba –revise widget.txt reva\widget.prt.1
ipemimport –u=infodba –p=infodba –revise widget.txt revb\widget.prt.3
ipemimport –u=infodba –p=infodba –revise widget.txt revc\widget.prt.8
```

In each case, the path to the model's file is specified on the command line. The model identified in the instruction file depends on the file that the utility loads.

Another simple instruction file is shown below.

```
# Sample ipemimport instruction file
D:\wwc\models
D:\wwc\bolts\bolt.prt.2 item_id=702283 item_desc="Bolt family generic"
  model_name=BOLT_A model_type=PRT item_id=702284
  model_name=BOLT_B model_type=PRT item_id=702285
  model_name=BOLT_C model_type=PRT item_id=702286
```

The first line is a comment. The second line specifies a directory D:\wwc\models. The ipemimport utility will load all Creo® models that it finds in that directory. The third line specifies a Creo® part file bolt.prt.2. Two optional arguments appear on this line to specify the model's item ID and, if the item has to be created, a description to include with the item.

In this example, suppose bolt.prt.2 contains a family table. The ipemimport utility automatically imports all family table instances, but in this case, the user wants to specify the item ID where each instance is to be stored, which is what the last three lines accomplish. Since the file name cannot identify the individual instances, the last three lines use the model_name and model_type keywords to identify these models.

The following instruction file specifies a multifield key for the vendor and the item name.

```
C:\work\widget.prt.5 item_type="ITI9_PropItem" item_id="DK_Widget1"
property[Item:ITI9_PropItem.iti9_name]="Widget 1"
property[Item:ITI9_PropItem.iti9_Vendor]="Widgets_R_Us"
```

The Item Type and Item ID are specified too.

The instruction file below shows the property keyword.

```
D:\Work_Dir\A2.asm property[Item:Ipm9_AllPropItem.ipm9_dateArray]="{15-Oct-2012
08:55,12-Jan-2012 05:14,25-Mar-2012 06:20}" "property[ItemRevision
Master:Ipm9_AllPropItemRevisionMaster.user_data_1]"="data1" "property[Item
Master.user_data_2]"="data2"
property[ItemRevision:Ipm9_AllPropItemRevision.ipm9_cascadingLOV]="Detroit"

D:\Work_Dir\P3.asm property[Item:Ipm9_AllPropItem.ipm9_dateArray]="{15-Oct-2012
12:32,12-Jan-2012 05:14,25-Mar-2012 06:20}" "property[ItemRevision
Master:Ipm9_AllPropItemRevisionMaster.user_data_1]"="data1" "property[Item
```

```
Master.user_data_2]"="data2"
property[ItemRevision:Ipm9_AllPropItemRevision.ipm9_cascadingLOV]="Boston"

D:\Work_Dir\P4.asm property[Item:Ipm9_AllPropItem.ipm9_dateArray]="{15-Oct-2012
14:55,12-Jan-2012 05:14,25-Mar-2012 06:20}" "property[ItemRevision
Master:Ipm9_AllPropItemRevisionMaster.user_data_1]"="data1" "property[Item
Master.user_data_2]"="data2"
property[ItemRevision:Ipm9_AllPropItemRevision.ipm9_cascadingLOV]="Mumbai"
```

Note

The ipemimport utility does not recognize or enforce the Creo® file version extensions specified in the instruction file. Creo® itself controls which file versions are opened.

Examples

The following example shows a simple use of this import utility.

```
ipemimport –u=infodba –p=infodba c:\ProeMdls
```

This command will load the latest version of each Creo® model from the `c:\ProeMdls` directory into Teamcenter, but it will skip any model that already has a dataset in Teamcenter. For the models that are imported, new items, item revisions, and datasets will be created as needed, attribute values will be copied to Teamcenter and BOMs will be created or updated.

To force all models to be imported, use the `–overwrite` option:

```
ipemimport –u=infodba –p=infodba -overwrite c:\ProeMdls
```

For items that already exist, the models will be imported into each item's latest revision, replacing Creo® model files that may have been imported previously.

If a new item revision should be created for those items that already exist, the import utility could be used this way:

```
ipemimport –u=infodba –p=infodba -revise c:\ProeMdls
```

This command carries out the same operations as in the previous example, including creation of new items for models that have not been imported previously, but new revisions are created for existing items.

Renaming Creo® Models in Teamcenter

Renaming models stored in Teamcenter is not recommended, because it is common that the process cannot be completed. There are two key requirements for renaming any model managed by Teamcenter:

- All revisions of the model must be renamed
- All parent model revisions must be updated to reference to the renamed model

These actions cannot be completed successfully if any of the models is checked out, access restricted, or will not open into Creo®. The utilities provided verify the checkouts and access restrictions but cannot verify if models can be opened into Creo®. If models cannot be opened, the renaming processes will fail and may require cleanup.

The renaming of models is usually desired because the filename is “smart”, in the sense that it contains key strings that are used by the organization to find models by some set of criteria. Rather than have smart filenames, it is recommended that you leverage Teamcenter to store meaningful identifiers in the Item Name and use Teamcenter-assigned values for both the item IDs and CAD model names. The Item Name can be changed at any time without affecting Teamcenter or the integration. The Item Name appears in all Teamcenter interfaces and all integration dialogs and can be used for searching data in Teamcenter. It is even possible to have a different Item Name for each Item Revision.

If renaming of models is required, we recommend that the scope of the renaming and associated updates to higher-level models be restricted to WIP data.

Renaming with ipemrename

The `ipemrename` utility renames Creo® models and their datasets in Teamcenter. This interactive utility will check out a model from Teamcenter, open it in Creo®, rename the model and save it back to Teamcenter. It also checks out, updates, and saves all of the Creo® models that use the renamed model, and optionally changes the item ID and the names of other objects to match the new name.

Usage

ipemrename [**-login**] [**-u**=*username* **-p**=*password* [**-g**=*group*]] [**-role**=*rolename*]

[**-project=***project_id*]

[**-help**]

-item=*itemid* **-type=***type*

[**-item_uid**=*UID*]

-old_name=*oldname* **-new_name=***newname*

[**-work_dir=***directory*]

[**-no_gui**]

[**-no_autorename**]
[**-autorename_item_id=**true|false]
[**-autorename_item_name=**true|false]
[**-autorename_other=**true|false]
[**-autorename_other_exact_length=**true|false]

[**-servername=***name*]
[**-progress_file=***outputfile*] [*inputfile*]

Description

Before running the `ipemrename` utility, all models that will be renamed and all of the models that use them must be checked into Teamcenter. The `ipemrename` utility will stop with an error if it finds that a model to be renamed, or a model that uses a renamed model, is already reserved. In addition, models that were once managed by Pro/PDM® may carry an internal lock that prevents Creo® from renaming them. See the section on **Renaming Models Imported from Pro/PDM®** if you suspect a model cannot be renamed because of such a lock.

A log files named `ipemrename.txt` and `client_socket.log` are always produced containing information about the actions carried out during the renaming process.

The arguments for `ipemrename` are described below.

-autorename_item_id=true | false
Specifies whether the item ID should be changed to match the model name (optional). If true, the item ID is changed to the new model name if the old item ID matches the old model name, ignoring case. If false, the item ID is not changed. Specifying `-no_autorename` also sets this option to false. This option can follow `-no_autorename` on the command line to override `-no_autorename`. If not specified, the default is given by the `iman.autorename.itemid` user preference (see the ipem.properties file in the installation directory for a description of user preferences).

-autorename_item_name=true | false
Specifies whether the item name should be changed to match the model name (optional). If true, the item name and item revision name are changed to the new model name if their old names match the old model name, ignoring case. If false, the names are not changed. Specifying `-no_autorename` also sets this option to false. This option can follow `-no_autorename` on the command line to override `-no_autorename`. If not specified, the default is given by the `iman.autorename.itemname` user preference (see the ipem.properties file in the installation directory for a description of user preferences).

-autorename_other=true | false
Specifies whether the names of other objects within the item or item revisions should be changed to match the model name (optional). Examples of “other” objects include forms, BOM views, and non-Creo® datasets and their named references, i.e. any Teamcenter object in an item or item revision that is not a Creo® dataset. When true, these other objects are renamed to the new model name if their old names match the old model name, ignoring case (the `-autorename_other_exact_length` option determines how the names are matched). When false, the names are not changed.
Specifying `-no_autorename` also sets this option to false. This option can follow `-no_autorename` on the command line to override `-no_autorename`. If not specified, the default is given by the `iman.autorename.other` user preference (see the ipem.properties file in the installation directory for a description of user preferences).

-autorename_other_exact_length=true | false
Specifies how to match names when deciding whether to rename other objects in

the item or item revisions (optional). When true, the name of a Teamcenter object matches the Creo® model name if both names match character-for-character, ignoring case, and are exactly the same length (for named references, the file name without extension is tested). When false, the names match if the object's name starts with the model name, ignoring case. For example, suppose the model's name is WIDGET. If `-autorename_other_exact_length` is true, an item master form named WIDGET will match the model name, but not an item revision master form named WIDGET/A. If `-autorename_other_exact_length` is false, both forms match the model's name. This option only affects the "other" objects in the item and item revisions when `-autorename_other` is true. Exact length is always used when matching the model name to the item ID, item name, or item revision name. If not specified, the default is given by the `iman.autorename.other.exactlength` user preference (see the ipem.properties file in the installation directory for a description of user preferences).

-g=*group* The Teamcenter group (optional). Ignored when `-login` is specified.

-help Prints usage information and quits (optional).

-item=*itemid*
The item ID of the model to be renamed.

-item_uid The Teamcenter UID of the Item. This optional argument is only needed when the Item is not uniquely identified solely by its Item ID.

-login Uses the Teamcenter login dialog. When this option is specified, the `-u`, `-p`, `-g`, and `-role` options are ignored. Use `-login` to avoid having to escape special characters that may be included in username, password, group, or role. Some special characters cannot be escaped and require the use of `-login`.

-new_name=*newname*
The new name for the Creo® model. Do not include the file extension.

-no_autorename
Turns off automatic renaming of the item, item revisions, and other objects (optional). This is the same as setting `-autorename_item_id`, `-autorename_item_name`, and `-autorename_other` to false.

-no_gui Disables the graphical user interaction (optional). Dialogs and prompts are not display. All alert messages are written to the log file and/or console window only.

-old_name=*oldname*
The old name for the Creo® model. Do not include the file extension.

-p=*password* The Teamcenter password. Ignored when `-login` is specified. Password may contain special characters as long as the password is enclosed in double quotes. Double quotes cannot be contained in the password string.

-progress_file=*outputfile*
The name of a text file that this program will create to record its progress (optional). If an error occurs, the program can be restarted using the file as the *inputfile* argument.

-project=*project_id*

Specifies the project to use when connecting to Teamcenter (optional).

-revise Causes a new revision to be created for each model imported into Teamcenter (optional). The default is to import into the latest revision, skipping any model if the latest revision already contains a version of the model.

-role=*role_name*

Specifies the role name to use when connecting to Teamcenter (Optional). Ignored when –login is specified.

-servername=*name*

The Teamcenter server name. If not specified, the default server is that of the first server in the `client_specific.properties` file, found in the Teamcenter Portal installation directory. The name should be one of specified by a property such as `DEFAULT_SERVER.NAME` or `HTTP_SERVER_1.NAME` in the `client_specific.properties`. Ignored when `-login` is specified.

-type=*type* The three-letter Creo® file extension that identifies the type of the model, such as ASM. Alternatively, this may also be the dataset type, such as ProAsm.

-u=*username* The user's Teamcenter login name. Ignored when `-login` is specified.

.

Suppressing the GUI

By default, the ipemrename utility runs in an interactive mode with a dialog that lists all the models involved in the rename. Additionally, the interactive mode displays alerts for the user to respond to. When –no_gui is specified, the dialog and all alerts are suppressed.

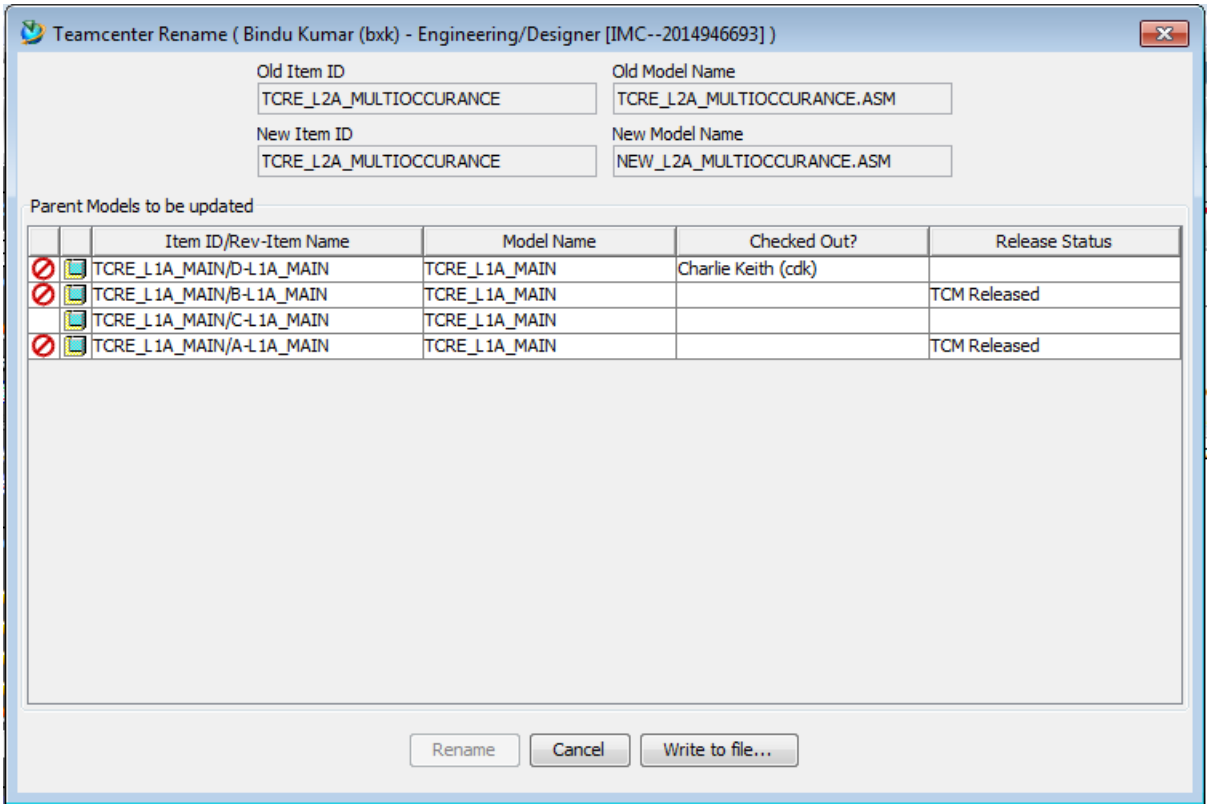


Figure 34 Dialog listing all Item Revisions involved in the rename.

The ⊘ symbol appears in the first column to indicate that a model cannot be checked out. A flyover for the icon provides a hint as to why the model cannot be checked out. The **Checked Out?** and **Release Status** columns also guide the user.

The **Rename** button remains disabled until first column does not contain the ⊘ symbol. Once the **Rename** button is enabled, the user can attempt to rename all revisions of the model and update all revisions of the parent models. Use the **Write to file…** button to obtain an HTML output of the dialog contents.

Examples

Suppose an assembly named “CYLINDER” is stored in an item whose item ID is 0000753, and it is decided to rename the model to match the item ID. The command to execute this change might look something like this:

```
ipemrename -u=infodba -p=infodba -item=0000753 \
   -type=ASM -old_name=CYLINDER -new_name=0000753 \
   -work_dir=tmp -progress_file=restart1.txt
```

All revisions of CYLINDER in item 0000753 are renamed to 0000753. This is recommended if there is a possibility that item 0000753 occurs in the imprecise BOMs of other assemblies. A directory named `tmp` is used as a temporary space for Creo® to work on these models, and a file named `restart1.txt` will be created in case the command aborts before all of the models are renamed.

Suppose an assembly that uses CYLINDER is currently checked-out by someone else. The `ipemrename` utility will stop with an error because it will not be able to check-out the other assembly, but this may occur after `ipemrename` has already renamed and saved several revisions. Since the `-progress_file` option is used in the example above, `ipemrename` can be restarted once the other assembly becomes available. The command to execute `ipemrename` is the same as before, but the output file created with the `-progress_file` option above is now given as an input to `ipemrename`:

```
ipemrename -u=infodba -p=infodba -item=0000753 \
   -type=ASM -old_name=CYLINDER -new_name=0000753 \
   -work_dir=tmp -progress_file=restart2.txt restart1.txt
```

Note

This command also creates a new text file, `restart2.txt`, which can be used to restart `ipemrename` again if another error occurs.

Suppose it is later decided to rename assembly 0000753 to 5000753 after a drawing named 0000753 has been added to the same item. The assembly and the drawing should both be renamed, but you must run `ipemrename` twice to do this. Either could be renamed first, but it may be desirable to avoid changing the item ID until both models are renamed. The two runs of `ipemrename` might look like this:

```
ipemrename -u=infodba -p=infodba -item=0000753 \
   -type=ASM -old_name=0000753 -new_name=5000753 -no_autorename \
   -work_dir=tmp -progress_file=restart.txt

ipemrename -u=infodba -p=infodba -item=0000753 \
   -type=DRW -old_name=0000753 -new_name=5000753 \
   -work_dir=tmp -progress_file=restart.txt
```

Notice that `-no_autorename` is used in the first run, when renaming the assembly. This prevents the item ID from changing and allows the second run to use almost the same arguments to rename the drawing, except for changing the `-type` argument and dropping `-no_autorename`. At the end of the second run, `ipemrename` will change the item ID to 5000753.

Renaming Creo® Models in Teamcenter with Creo®→Rename

The **Creo®→Rename** function is disabled OOTB for all users. Due to the challenges of renaming models in Teamcenter, it is recommended that renaming only be done by administrators. To enable **Creo®→Rename**, set the Teamcenter preference `Pro2_deferred_rename_enable` to `true`, as described elsewhere in this document. **Creo®→Rename** is a simplified way of executing the ipemrename utility.

Select a Pro* dataset in the Teamcenter Rich Client, then select **Creo®→Rename** to invoke the rename dialog. The New Model Name must be specified. The three options correspond to similar autorename preferences on the ipemrename command line.

- Apply to Item Name: set autrename_item_name=true when ipemrename is invoked
- Apply to Item ID: set autorename_item_id=true when ipemrename is invoked
- Apply to Other: set autorename_other=true when ipemrename is invoked

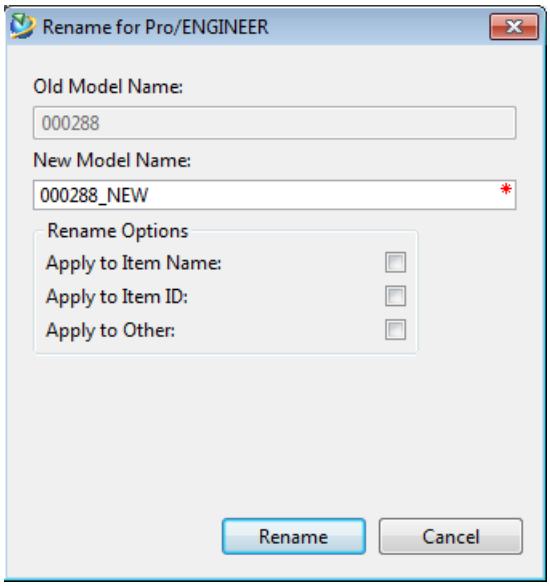


Figure 35 Teamcenter Rename dialog

Upgrading the Teamcenter Database

As new features are added to the integration, data stored in Teamcenter may require updating in order to take advantage of those new features. The ipem_upgrade_database utility is provided for this purpose. The utility is installed on the Teamcenter server in the **TC_ROOT\bin** directory. It will be found in the path when a Teamcenter command line is started. The utility must be executed on all Teamcenter servers. On UNIX, you may need to change the permissions on the file to run as execute.

Whether to run this utility, and which options to use, will depend on each customer's history with the Teamcenter Integration for Creo®. The following is not necessarily an exhaustive list of scenarios but is provided as a set of high-level guidelines.

If	Then	Why
Version 9.1.0 or later is your first deployment of the Creo® Integration	Do not run the utility	There is no pre-existing data to be scanned or updated
You first deployed the integration on a version earlier than 9.1.0	Run the utility with: -operation=pro2_instance	Family tables last saved in earlier versions of the integration will lack the Pro2_instance relation introduced in version 9.1.0. This relation is important for multi-site data sharing, as well as for potential enhancements and performance optimizations in future releases.

Usage

ipem_upgrade_database –u=*user* **–p**=*password*

-operation={deferred_rename | find_deferred_renames | fix_deferred_rename | pro2_instance}

[**-statistics**]

[**-dryrun**]

[**-debug**]

[**-help**]

Description

The `ipem_upgrade_database` utility is used to upgrade the Teamcenter database for new integration functionality. The utility has operations that can be performed one at a time. The typical steps in using the utility are the following:

1. Select an operation to run
2. Run the operation with –dryrun to determine if there is data that requires upgrading, or if there will be problems upgrading the database.
3. If data requires upgrading, then run the operation
4. If there are failures, fix the reported database problems and run again.

A user with dba permissions, such as infodba, must run the utility. Failures can result if the user does not have access to all data. This process must be run at all sites to ensure proper synchronization of data. Once an operation has been executed completely at a site, it should never have to be run again.

Note

If the following error occurs, make sure the path to ipem_upgrade_database is in the path variable.

Ipem_upgrade_database: command not found

Failed to process one or more records in batch file line numbers 1-5000 further processing halted.

The arguments for `ipem_database_upgrade` are described below.

`-debug`

An optional argument to specify that a detailed diagnostics debug file named **ipem_upgrade_database.debug** should be written to the working directory. This generates a very large file. It should only be used if error messages are encountered during a –dryrun execution.

`-dryrun`

An optional argument. When specified, the utility does not modify the database. This option is used to simulate the operation before live execution.

`-help`

Prints the help screen for the utility.

`-operation={deferred_rename | find_deferred_renames | fix_deferred_rename | pro2_instance}`

Required argument to specify the operation to be performed. The operation must be one of the following:

deferred_rename	Adds the required information to support **Teamcenter Duplicate** and **Creo®→Rename.** This operation should not be used unless advised to do so, by Siemens support or services.
find_deferred_renames	Writes information about models with deferred rename conditions into the log file. If any deferred rename conditions are listed, the parent models should be opened into Creo® and saved back to Teamcenter, to remove the deferred rename conditions and enable the site to set `checkout.deferredrename.enable=false` in ipem.properties. These preferences can improve the performance of all integration load operations.

	This operation should not be used unless advised to do so, by Siemens support or services.
fix_deferred_rename	Identifies and fixes deferred rename data that exists but is corrupted. This operation should not be used unless advised to do so, by Siemens support or services.
pro2_instance	Adds Pro2_instance relations to enable improved multi-site data sharing of Creo® family tables.

Note

The **deferred_rename**, **find_deferred_renames** and **fix_deferred renames** operations are maintained for specialized purposes related to certain legacy data conditions, and should not be used unless explicitly advised to do so. In most cases, only the **pro2_instance** operation will be needed.

-p=*password*

The user's Teamcenter password. On Windows, the password may contain special characters as long as the password is enclosed in double quotes. Double quotes cannot be contained in the password string. On Unix the password may contain special characters as long as they are each escaped with backslash character. Unix passwords may not contain the characters $ { } " (double quote) ` (grave accent)

-statistics

An optional argument, which will run the utility and output to the log file the count of each type of GRM relation this utility queries. No other processing will be done when this option is specified. Run this operation prior to using –dryrun, to understand the size of the operation. This will execute quickly. This option is only meaningful for the **deferred_rename** and **pro2_instance** operations.

-u=*username*

The user's Teamcenter login name.

Fixing missing deferred rename data

There are times when duplicated structures cannot be opened from Teamcenter. This can occur because the deferred rename data was corrupt in the original structure. The –`fix_deferred_rename` operation can be used to fix the deferred_rename data of the original structure. The deferred rename data of the replica structure cannot be repaired, and it may not be possible to open the assembly into Creo® without errors that must be fixed manually.

Similarly, there are times when renamed data cannot be opened from Teamcenter. This can occur because the deferred rename data was corrupt on relations between the parent and renamed child models. There is no way to fix the deferred rename data after the rename operation.

Preventive maintenance is the best way to avoid these problems. Use the `-fix_deferred_rename` and `-dryrun` options to determine whether the database has problems, or simply run without the -

`dryrun` option, to fix any issues. Use the following command line to determine if corrupt deferred_rename data exists in the database:

```
ipem_upgrade_database -u=<username> -p=<password>
–operation=fix_deferred_rename –dryrun
```

The following command line will fix bad user data.

```
ipem_upgrade_database -u=<username> -p=<password>
–operation=fix_deferred_rename
```

Upgrade family tables for Multi-site

Upgrading family tables is only required for sites with Creo® data that was last saved with integration version 8.6.2 or earlier. The family table upgrade process adds **Pro2_instance** relations to the database. The **Pro2_instance** relations lead from each generic dataset to the item revisions of all instances belonging to the generic's family table. Teamcenter multi-site can then be configured to traverse the **Pro2_instance** relations during data sharing operations, enabling the transfer of the entire family table as a whole. Starting with version 9.1.0 of the integration, the **Pro2_instance** relations are created during all save and import operations. To add the **Pro2_instance** relations for pre-existing data, use the

`-operation=pro2_instance` option.

The following command line will run the **ipem_upgrade_database** utility in dry run mode to determine whether there will be any problems creating **Pro2_instance** relations. The `–dryrun` option will not create any relations and will only record problems in the log file.

```
ipem_upgrade_database –u=<username> -p=<password> –operation=pro2_instance
–dryrun
```

Once ready to create the Pro2_instance relations, the following command line may be used:

```
ipem_upgrade_database –u=<username> -p=<password> –operation=pro2_instance
```

Chapter 5 Data Model

Integration Relations loaded by the BMIDE Template

The integration's BMIDE template loads relation types into the database. During save and ipemimport operations, the integration creates these relations between models. The relation types created between models are derived from Creo®'s internal references. The following table contains the relations created by the integration.

Display Name	Relation Type	Description
Creo dependency	IPEM _dependency	Prior to version 9.1.0 this was the primary relation created by the integration. Starting with IPEM 9.1.0, this relation is no longer created. It continues to be supported by integration actions that read the relations in Teamcenter. This is a general relation that was used to represent most PTC references. In IPEM 9.1.0 it was replaced by the many Pro2_* relations listed below.
Creo mandatory dependency	IPEM_mandatory_dependency	A relation that is not used.
Creo generic	IPEM_master_dependency	A relation from a family table instance dataset to the item revision of the instance's generic model.
Creo suppressed membership	IPEM_suppressed_dependency	A relation from an assembly dataset to the item revisions of suppressed child components. This relation is only created if all occurrences of the component are suppressed;

		otherwise, a Pro2_membership relation will exist. In the Teamcenter BOM, suppressed occurrences are tracked using the bl_occ_suppressed attribute.
Creo bulk membership	Pro2_bulk	Relation between assemblies and bulk parts. Bulk parts are included in the Teamcenter BOM. Bulk relations are included by the **Required** filter, during **Open** from Teamcenter.
Creo diagram item	Pro2_diagram	Relations from diagrams to drawings, formats, notebooks, and other diagrams. This relation is included by the **Required** filter.
Creo drawing item	Pro2_drawing_model	A relation from a drawing to a part or assembly referenced by the drawing. This relation is included by the **Required** filter.
Creo external SREP membership	Pro2_external_srep	A relation from an external simplified representation assembly to its master assembly. This relation is included by the **Required** filter.
Creo interchange assembly	Pro2_interchange_assembly	A relation from an interchangeable member to its interchange assembly.
Creo interchange membership	Pro2_interchange_membership	A relation from an interchange assembly to its interchangeable members. This relation is included by the **Required** filter.
Creo notebook (was called layout prior to Creo® 2.0. See CEM below)	Pro2_layout	A relation to a notebook.
Creo markup item	Pro2_markup	A relation from a markup to one of its dependents. This is included by the **Required** filter.
Creo membership	Pro2_membership	A relation from an assembly to one of its child components. Membership relations are included in the Teamcenter BOM and are included by the **Required** filter.

Creo merge dependency	Pro2_merge	A relation from a part or assembly to a merge part or merge assembly. This relation is included by the **Required** filter.
Creo mfg dependency	Pro2_mfg_reference	A relation between assemblies and manufacturing items. This relation is included by the **Required** filter. This relation is only created for Wildfire 4.0. For Wildfire 5.0, Creo® Parametric 1.0, and Creo® Parametric 2.0 manufacturing files are internal to their assemblies, and so a relation is not created.
Creo skeleton model	Pro2_skeleton	A relation to a skeleton model. It is included by the **Required** filter.
Creo substitute dependency	Pro2_substitute	A relation created when an assembly references a substitute model. This relation is included by the **Required** filter.
Creo report item	Pro2_report	A relation from a report dataset to one of its dependencies. This relation is included by the **Required** filter.
Creo reference	Pro2_reference	A relation from a part or assembly to a non-required model. This is a catch-all for non-required references that are not handled by other relations in this table. For example, this relation gets created for ghost relationships and copied geometry references in Creo®.
Creo UDF	Pro2_udf	This relation is created for assemblies or parts that reference UDF (.gph) models. This relation is included by the **Required** filter.
Creo UDF source model	Pro2_udf_source	This relation represents the Creo® reference from a UDF model to its source model. The relation is included by the **Required** filter.
Creo UDF reference model	Pro2_udf_reference	This relation represents the Creo® relationship from a UDF model to its reference model.

Creo cem	Pro2_cem	This relation links a Creo® model to its layouts
Creo format dependency	Pro2_format	This relation links a Creo® drawing to its format.
Creo instance	Pro2_instance	A relation between a Creo® generic part or assembly and each of its family table members.
Creo BOM exclusion	Pro2_bom_exclusion	A relation between a Creo® assembly dataset and a component dataset, when all occurrences of the component have been excluded from the assembly BOM with the **RMB→Teamcenter→Exclude from BOM** feature. If all occurrences of a component are both Suppressed and Excluded from the BOM, an IPEM_suppressed_dependency is created while saving, not a Pro2_bom_exclusion. If one occurrence of a component is excluded from the BOM and another occurrence is suppressed, the Pro2_bom_exclusion relation is created.

Chapter 6 Upgrading Creo Versions

When to reinstall the integration

The installation DLLs are specific to the major Creo® version. For example, if the integration was installed with Creo® 7.0.4.0, it can be used as-is with Creo® 7.0.5.0. But the integration must be reinstalled if Creo® is upgraded to Creo® 8.0.5.0. Failing to upgrade the integration will cause it to fail to load.

Chapter 7 Troubleshooting

Reporting problems

Whenever reporting an issue to Support Center, please include the following information:

- Version numbers of all involved software. At a minimum, this must include Teamcenter, Creo®, and the Teamcenter Integration for Creo®. If the problem is related to JT translation, we will also need the exact version number of the CreoToJT translator.
- A complete problem description, including detailed steps for reproducing it.
- The integration's `txdlog.txt log` file. Before capturing the log file, be sure to set the **log.level** preference, in ipem.properties, to a value of **TRACE**. This will produce the most verbose logging available.
- Each integration session will produce a **client_socket.log** file. Before capturing the log file, be sure to Set **IPEM_LOG_DEBUG_PRINTS=1** in **ipemenv.bat** to increase the level of detail in the **client_socket.log** file. This file is written in the integration start directory on each client. The directory is the **IPEM_START_DIR** in ipemenv.bat. Set to %USER_PROFILE% by default.
- The utilities such as **ipemimport**, **ipemrename** and the dispatcher processes for auxiliary file (JT, DXF, …) generation Teamcenter duplicate resolution put the **client_socket.log** file into the folder where they are run. For dispatcher processes the **client_socket.log** is written to the stagging directory.st
- The **integration** session's tcserver syslog file. This is **not** the same as the Teamcenter Rich Client's syslog file. You can positively identify both the location and the name of the integration's syslog file by searching in the txdlog.txt file for the string 'syslog'. For example:

```
2837  <nsl:extraInfo key="displayCurrentCountryPage" value="false"></nsl:extraInfo>
2838  <nsl:extraInfo key="hostName" value="cii6p942"></nsl:extraInfo>
2839  <nsl:extraInfo key="locationCodePref" value=""></nsl:extraInfo>
2840  <nsl:extraInfo key="syslogFile" value="c:\temp\tcserver.exe2f8cf63.syslog"></nsl:extraInfo>
```

Figure 36 - Location of the tcserver syslog

- If the problem involves Dispatcher-based translation, we will need the `ipemaux.txt` log file. This file will be found in the configured integration's startup directory, as specified by the value of `%IPEM_START_DIR%` in the ipemenv.bat file. The ipemaux.txt file will have a suffix that matches the Dispatcher tasks's ID, for example `ipemaux_U2917760ec5d3b2b842291.txt`.
- If the problem involves launching, or sending Creo® data, from either the Teamcenter Rich Client or Active Workspace, include all log files found under `%USERPROFILE%\AppData\Local\ipem`.

- If Creo® does not start or crashes, capture the std.out file written by Creo® to the `%USERPROFILE%` folder.
- If your issue is related to server-based post actions, such as **Revise** or **Save As**, the integration's server component can be configured to log its activity by setting a global **TXD_DEBUG** environment variable to true.

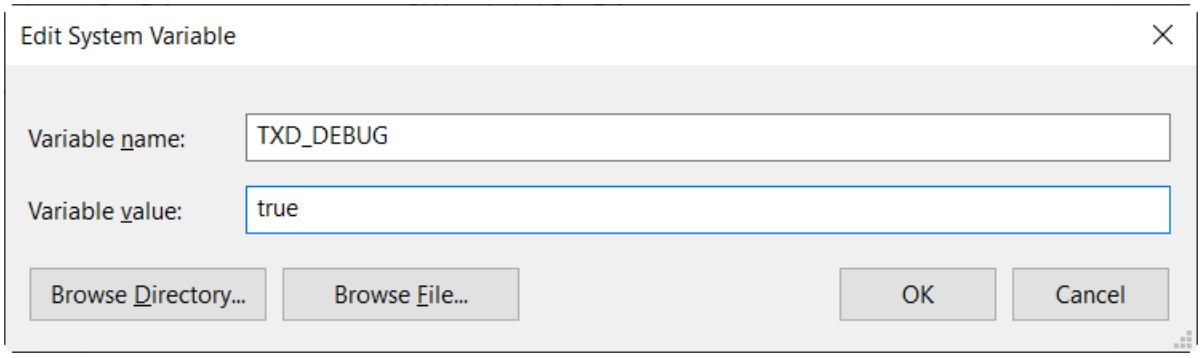


Figure 37 - TXD_DEBUG environment variable

When **TXD_DEBUG** is set as shown above, integration messages will be written to the Teamcenter session's tcserver syslog. These messages are prefaced with either **LIBTXD_DEBUG** or **LIBTXD_ERROR** keywords, depending upon the nature of the message.

- If you are troubleshooting a performance issue, include both the server and the client performance journal (pjl) files. Server journaling is described in the Teamcenter documentation; see Performance troubleshooting for instructions on client-side journaling.

Failure to start or connect to Creo®

Command line utilities

The `ipemimport, ipemexport` and `ipemrename` utilities use a non-graphical Creo® session to gather information about the models to be imported. If these utilities cannot start Creo®, the following points may be helpful.

- Make sure Creo®'s bin directory is in your PATH environment variable.
- Make sure Creo® does not report any errors or warnings when it starts. Such messages could interfere with the utility. You must be able to start an interactive session of Creo® using the `bin\startcreo.bat` command, found in the Teamcenter Integration for Creo® installation directory, without seeing any messages from Creo® that require a response from the user.
- When running any of the three utility programs, the current working directory must not contain a copy of protk.dat, creotk.dat, or any other file that attempts to load a synchronous Pro/Toolkit application. This means you should not run the utilities from the integration's installation

directory, because this directory will always contain a copy of creotk.dat. Change to another directory, such as c:\temp, before running the utility program.

From Teamcenter

Starting Creo® from Teamcenter

The Teamcenter Rich Client ordinarily detects when a Creo® session is active and sends the selected data directly to the integration's **Open** dialog.

If, instead of reusing the existing Creo® session, the integration starts a second session, a possible reason is that the task list function cannot be found and executed. The executable file name is "tasklist.exe". This system executable must be found in the user's path.

If the existing Creo® session was started without the integration, the send of data would fail without messages. The existing session must be started as an integration Creo® session, the user does not have to login before the send from the Teamcenter Rich Client.

Exporting data from Teamcenter

When a user attempts to open models from the Teamcenter Rich Client but gets instead gets an error message which states

```
"Launch to Session Failed due to invalid or no selection"
```

the failure might be caused by an invalid value for the Teamcenter option `TC_check_out_dir`. The Rich Client plugin provided by the Teamcenter Integration for Creo® creates a file named `from_portal.txt` in the directory that is found by the following search order:

1. `TC_check_out_dir` A Teamcenter preference
2. `user.dir` A user environment variable

Verify that these values are set correctly.

Failure to load the Teamcenter Integration for Creo®

A successful startup results in a Teamcenter menu appearing in the Creo® session:

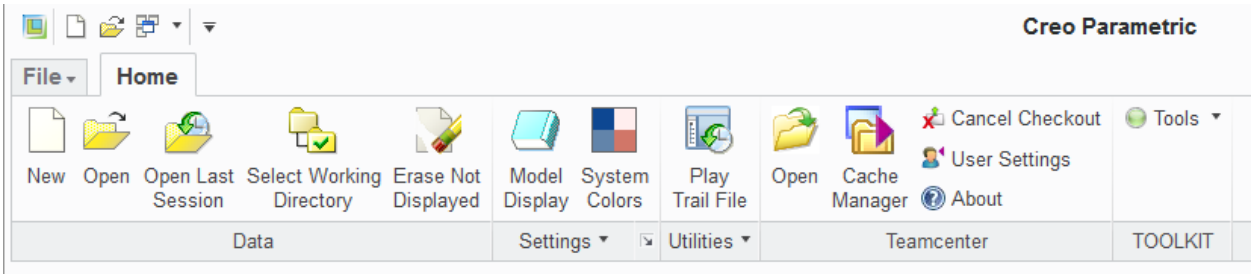


Figure 38 - Teamcenter menu in Creo®

If the integration is registered in multiple files Creo® .dat files, Creo® will issue and 'ipem' is already registered Creo Toolkit startup error. 'ipem' must only be registered in the creotk.dat file in `%IPEM_START_DIR%`. It cannot be registered in a protk.dat file in the same folder. Eliminate the ipem section from the protk.dat or eliminate the protk.dat all together.

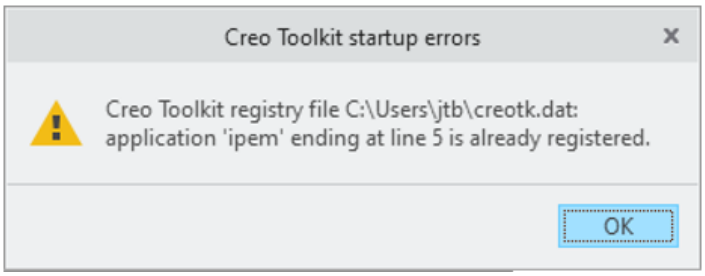


Figure 39 – Startup failure due to misconfiguration

If the runtime environment is misconfigured in any way, the message 'Foreign program "ipem" failed to initialize: error code 73' may be displayed by Creo®. It will likely be followed by a second message 'Startup of application "ipem" failed' message.

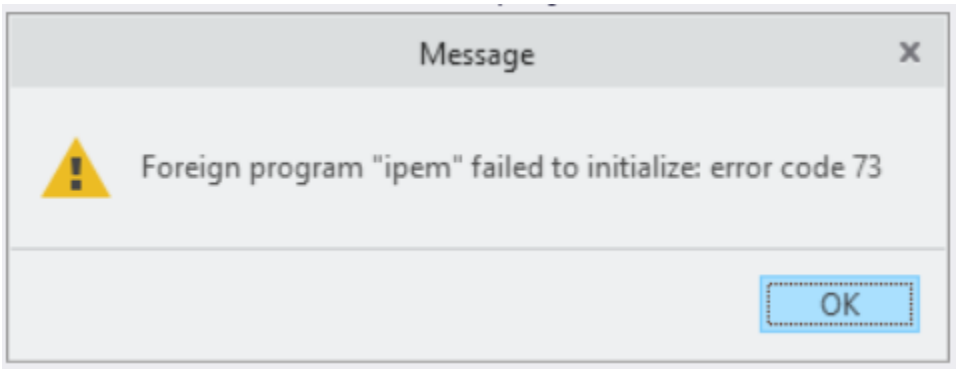


Figure 40 – Startup failure due to misconfiguration

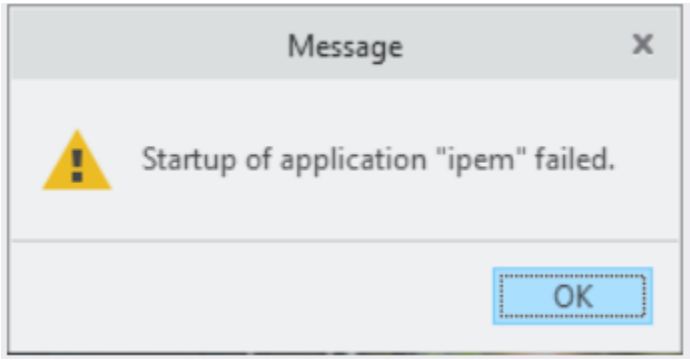


Figure 41 – Startup failure due to misconfiguration

If the above alerts occur, check the client_socket.log found in `%IPEM_START_DIR%` . If there is a problem with the java bridge, the following timeout will be seen:

```
(00:00:00:066) [INFO] - Initializing connection to the Teamcenter bridge
(00:00:00:066) [INFO] - Waiting for the Teamcenter bridge to startup (30 second(s) left)
(00:00:01:070) [INFO] - Waiting for the Teamcenter bridge to startup (29 second(s) left)
(00:00:02:076) [INFO] - Waiting for the Teamcenter bridge to startup (28 second(s) left)
(00:00:03:090) [INFO] - Waiting for the Teamcenter bridge to startup (27 second(s) left)
(00:00:04:092) [INFO] - Waiting for the Teamcenter bridge to startup (26 second(s) left)
(00:00:05:103) [INFO] - Waiting for the Teamcenter bridge to startup (25 second(s) left)
(00:00:06:115) [INFO] - Waiting for the Teamcenter bridge to startup (24 second(s) left)
(00:00:07:116) [INFO] - Waiting for the Teamcenter bridge to startup (23 second(s) left)
(00:00:08:126) [INFO] - Waiting for the Teamcenter bridge to startup (22 second(s) left)
(00:00:09:136) [INFO] - Waiting for the Teamcenter bridge to startup (21 second(s) left)
(00:00:10:150) [INFO] - Waiting for the Teamcenter bridge to startup (20 second(s) left)
(00:00:11:163) [INFO] - Waiting for the Teamcenter bridge to startup (19 second(s) left)
(00:00:12:178) [INFO] - Waiting for the Teamcenter bridge to startup (18 second(s) left)
(00:00:13:193) [INFO] - Waiting for the Teamcenter bridge to startup (17 second(s) left)
(00:00:14:201) [INFO] - Waiting for the Teamcenter bridge to startup (16 second(s) left)
(00:00:15:211) [INFO] - Waiting for the Teamcenter bridge to startup (15 second(s) left)
(00:00:16:226) [INFO] - Waiting for the Teamcenter bridge to startup (14 second(s) left)
(00:00:17:240) [INFO] - Waiting for the Teamcenter bridge to startup (13 second(s) left)
(00:00:18:255) [INFO] - Waiting for the Teamcenter bridge to startup (12 second(s) left)
(00:00:19:267) [INFO] - Waiting for the Teamcenter bridge to startup (11 second(s) left)
(00:00:20:277) [INFO] - Waiting for the Teamcenter bridge to startup (10 second(s) left)
(00:00:21:292) [INFO] - Waiting for the Teamcenter bridge to startup (9 second(s) left)
(00:00:22:305) [INFO] - Waiting for the Teamcenter bridge to startup (8 second(s) left)
(00:00:23:309) [INFO] - Waiting for the Teamcenter bridge to startup (7 second(s) left)
(00:00:24:323) [INFO] - Waiting for the Teamcenter bridge to startup (6 second(s) left)
(00:00:25:337) [INFO] - Waiting for the Teamcenter bridge to startup (5 second(s) left)
(00:00:26:347) [INFO] - Waiting for the Teamcenter bridge to startup (4 second(s) left)
(00:00:27:361) [INFO] - Waiting for the Teamcenter bridge to startup (3 second(s) left)
(00:00:28:370) [INFO] - Waiting for the Teamcenter bridge to startup (2 second(s) left)
(00:00:29:384) [INFO] - Waiting for the Teamcenter bridge to startup (1 second(s) left)
(00:00:30:395) [ERROR] - Could not connect to the Teamcenter bridge
(00:00:30:395) [ERROR] - There could be a problem with the Teamcenter bridge java application
(00:00:30:395) [ERROR] - Exception: Bridge connection timeout reached
```

Figure 42 – Bridge failure messages in client_socket.log file.

Check the `%IPEM_START_DIR%\ipemenv.bat %IPEM_JRE%` path is correctly set. Check the Installation Guide for the required JRE.

The resulting log file may contain details about what has gone wrong.

A very common reason for lack of a Teamcenter menu is installing the integration on a 64-bit Windows client, without redirecting the installer to a 64-bit JRE which is compatible with the Teamcenter FMS client libraries. It is usually best to use the same JRE that is used by the Teamcenter Rich Client.

- Under normal conditions, Creo® loads the Teamcenter menu by reading the `creotk.dat` file that is found in the current working directory, which is typically %USERPROFILE%. In some cases, Creo® might find and read a `creotk.dat` file from a different location. Search your Creo® installation for another `creotk.dat` file. Also, another `creotk.dat` file might be specified by the `creotk.dat` preference in a Creo® configuration file such as config.pro. If another `creotk.dat` file exists in one of these locations, paste the lines from %USERPROFILE%\`creotk.dat` into that `creotk.dat` file.
- The installation DLLs are specific to the major Creo® version. For example, if the integration was installed with Creo® 7.0.4.0, it can be used as-is with Creo® 7.0.5.0. But the integration must be reinstalled if Creo® is upgraded to Creo® 8.0.5.0. Be sure the proper Creo® version was used during the installation. Failing to upgrade the integration will cause it to fail to load.
- Use only mapped drives in paths. Do not use UNC paths. For example, the path `C:\shared\ugs\ipem` should cause no problems, but a UNC path such as `\\mycomputer\shared\ugs\ipem` should be avoided.
- The integration reads its startup parameters from the %IPEM_DIR%\ipemenv.bat file. Several key parameters may be overridden by system or user-level environment variables. Verify that none of the following exists as an environment variable, or, if it does, that it has the appropriate value:
    - IPR_DIR
    - FMS_HOME
    - IPEM_JRE
    - PROE_DIR
    - IPEM_PROE_CMD
    - PROE_COMMON_DIR
    - IPEM_DIR
    - IPEM_SOA_CLIENT_DIR
    - IPEM_START_DIR
- Make sure the bin subdirectory in the Creo® installation is specified in the user's **PATH** environment variable, and that no other versions of the Creo® bin directory precede it in the **PATH** definition.
- Sometimes the Teamcenter tab will not display in Creo®'s ribbon bar. This can be caused by customized model type-specific menu layouts. To fix this issue, perform the following steps:
    - Set the Creo® preference shown below and restart Creo®:

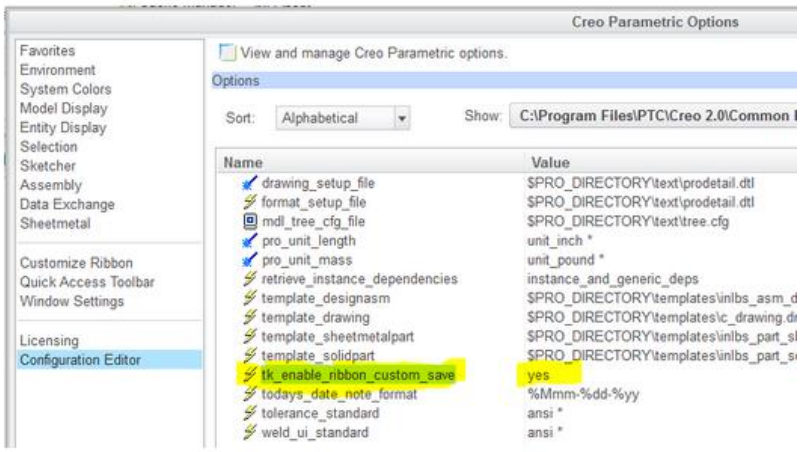


Figure 43 Creo® config.pro setting to save ribbon file

- Open or create a model of the type associated with the menu customization, then select the **Right Mouse Button** (RMB) anywhere on any ribbon:

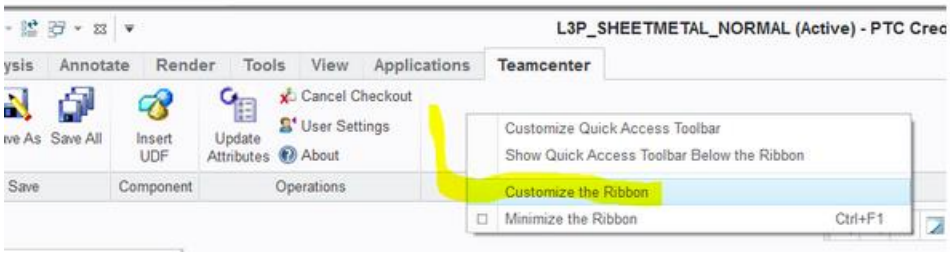


Figure 44 Customize the Ribbon

- Find the location on the right panel where you want to add the “Teamcenter” menus. Below is an example:

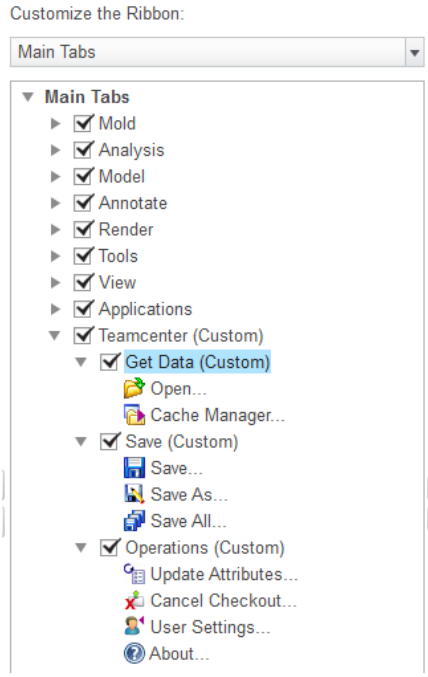


Figure 45 Modify Ribbon

- Make sure to look for the menu commands under the TOOLKIT Commands. Use the **Add/Remove** button as needed:

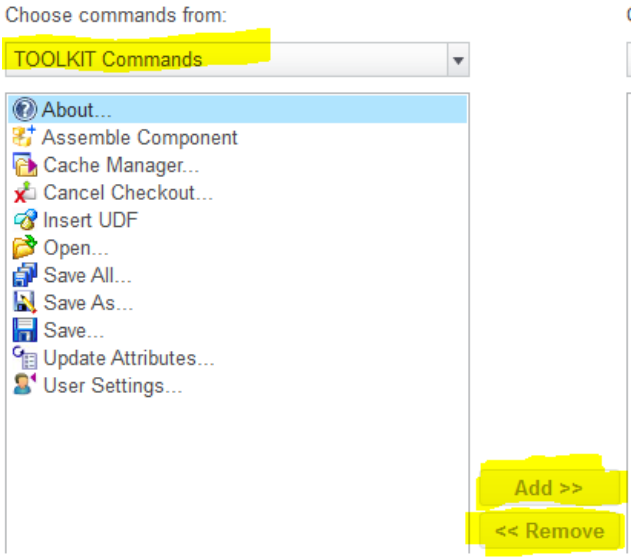


Figure 46 Select Toolkit Commands

- Repeat as necessary for each model type with issues.
- When finished press the Import/Export button near the bottom of the screen, then select **Save the Auxiliary Application User Interface**.

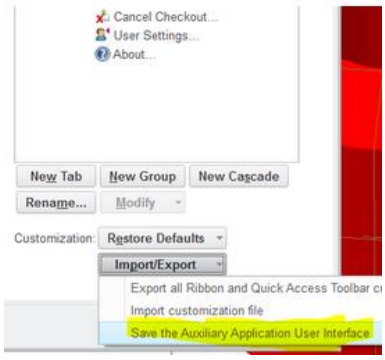


Figure 47 save the ribbon file

If that option is not shown, then the *tk_enable_ribbon_custom_save* preference was not set to true (see above **Creo® settings**)

- Find the saved file created from the previous step:

```
%IPEM_DIR%/text/toolkitribbonui.rbn
```

- Rename that file to `ipem_creo4.rbn` (or `ipem_creo4_aw.rbn` if Active Workspace is enabled) and copy it to the following locations, overwriting any existing file with the same name:

```
%IPEM_DIR%\text\ribbons
%IPEM_DIR%\text\usascii\ribbons
%IPEM_DIR%\text\chinese_cn\ribbons
%IPEM_DIR%\text\chinese_tw\ribbons
%IPEM_DIR%\text\french\ribbons
%IPEM_DIR%\text\german\ribbons
%IPEM_DIR%\text\italian\ribbons
%IPEM_DIR%\text\japanese\ribbons
%IPEM_DIR%\text\korean\ribbons
%IPEM_DIR%\text\russian\ribbons
%IPEM_DIR%\text\spanish\ribbons
```

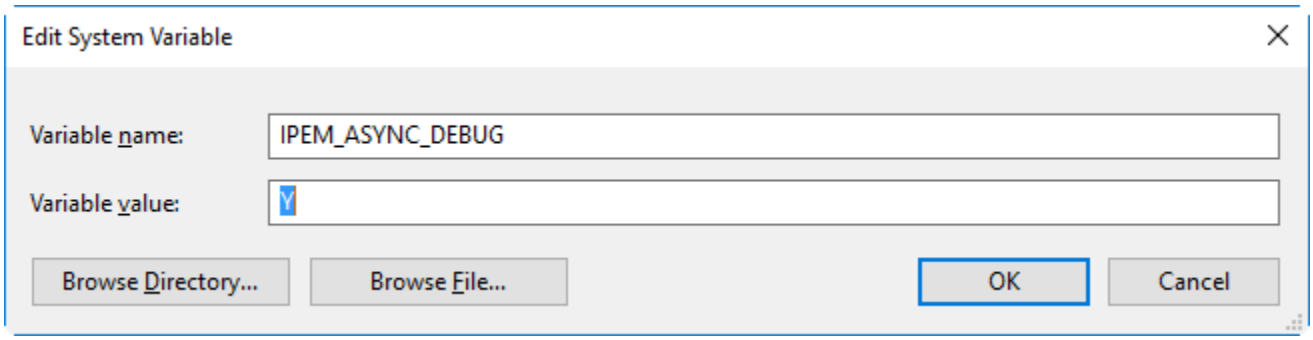
- End the Creo® session.
- Creo® keeps a cached file with the changes you just made. To ensure that the cached file does not interfere with the changes you just made; please delete the following file:

```
%USER_PROFILE%\AppData\Roaming\PTC\ProENGINEER\Wildfi
re\.wf\.Settings\creo_parametric_customization.ui
```

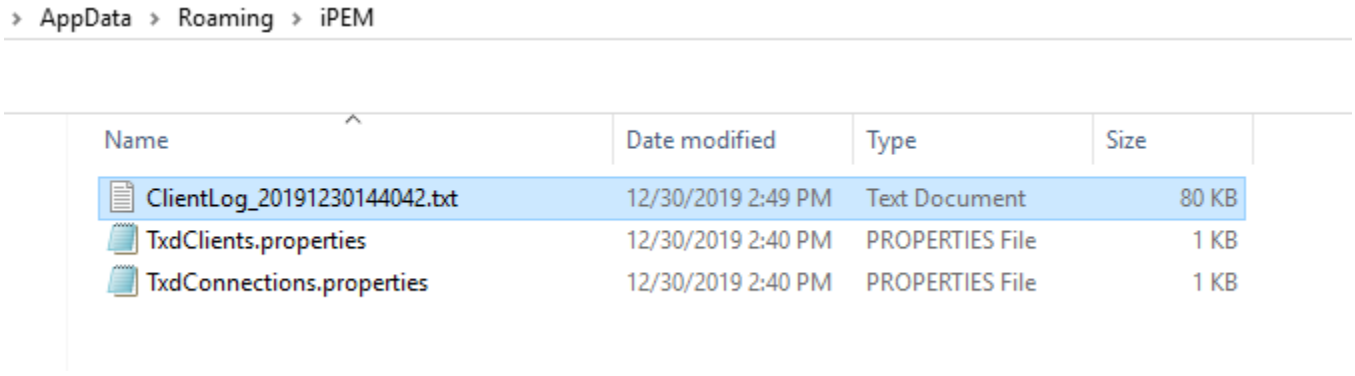
- Start Creo® using the integration launch command, and verify that the Teamcenter menu is now loaded into the Creo® ribbon bar.

Cache Manager launch issues

The integration's **Cache Manager** may fail to start if unexpected environmental factors prevent it from identifying the running Creo® process (xtop.exe). Set the following environment variable to activate a detailed **Cache Manager** log file:



The log file will be located at %APPDATA%\Roaming\iPEM, and will be named **ClientLog_*datetime*.txt**, for example:



Troubleshooting runtime issues

- Often problems must be sent to Siemens for analysis. We will need the **txdlog.txt** and **client_socket.log** files to better understand the issue. To generate the **txdlog.txt** file for a particular problem, open **ipem.properties** and add the following lines to the end (this will ensure that it overwrites any other setting) or uncomment and edit the existing lines:

  ```
  log.enable = true
  log.file = c:\\temp\\txdlog.txt
  log.level = TRACE
  ```

  To generate the **client_socket.log** file, with full details, edit the **ipemenv.bat** file and change the value of **IPEM_LOG_DEBUG_PRINTS** from **0** to **1**.

- Restart Creo® and rerun the test. Send us the **txdlog.txt** file which will be in the directory specified by the log.file setting, as in the example above. Send the **client_socket.log** file, which will be found in the **%USERPROFILE%** directory. Also, send the **tcserver syslog** that corresponds to the integration session. The server host and full path to the syslog file are recorded in the **txdlog.txt** file and can be found by searching for the string "syslog".

- Sometimes problems cannot be deliberately or consistently reproduced, but instead happen unpredictably. By default, the integration will overwrite its txd log file each time a new session is started but can also be configured to preserve each log file. This is done by setting `log.append.date = true` in ipem.properties. Similarly, the **client_socket.log** file is overwritten with each new session, unless `IPEM_LOG_APPEND_DATE` is changed from 0 to 1, in the **ipemenv.bat** file.
- For issues in which Creo® crashes or displays an error of some kind, it's important to locate and send the Creo® trail file generated by the session. The Creo® trail file, by default, is found in the session's current working directory.
- For issues in which Creo® crashes or displayes an error of some kind, validate that all config.pro settings are set according to the PTC configuration settings portion of this document.
- Customers working with extremely large assembly structures may encounter java "out of memory" errors. If so, the maximum java heap size may be increased by editing the ipemrunnersoa.bat file, in the integration's client installation directory, and increasing the **–Xmx** value. For example, change this:

  ```
  set ARGS=%ARGS% -Xms128m -Xmx4096m
  ```

  To this:

  ```
  set ARGS=%ARGS% -Xms128m -Xmx8192m
  ```

- It may be necessary to specify a specific language supported by Teamcenter that is not the native language of the client operating system. For example, if the client native operating system is Hungarian and the desired and supported Teamcenter language is German. To override the native operating system client value, edit the %IPEM_DIR%\ipemrunnersoa.bat and insert a new line immediately following this line:

  ```
  set ARGS=%ARGS% -DTXD_USE_SOA=1
  ```

  The new line specifies the language and country-specific locale you want the integration to use. For example

  ```
  set ARGS=%ARGS% -Duser.language=xx -Duser.country=YY
  ```

  Where xx is the language (e.g., "en" for English), and YY is the country (e.g., "US" for United States).

- Use of the JAVA_TOOL_OPTIONS environment variable to launch Java programming language agents is not supported. Such agents have been found to disrupt **Teamcenter Integration for Creo®** functionality. Such agents should be disabled in the integration environment by unsetting this variable in the integration runtime environment.
- Integration dialogs are hidden behind Creo®. When Creo® is started and the integration loads a program called txdjavaontop.exe. Make sure that the program is running when the dialogs are on top. If the program is not running, try to determine why not. User can pop windows to the top using **Alt-Tab**. Another option is to configure the `window.alwaysOnTop = true` in ipem.properties. Doing so forces all dialogs and alerts to the very top of the window stack.

- During Save operations, getting Teamcenter Alert: “The active model has pending operations. Please complete these operations before attempting Teamcenter actions.” This alert happens when folders are restricted to the user. Be sure that config.pro settings for `save_object_in_current` and `override_store_back` are set according to PTC configuration settings chapter within this document.

Send to Creo® from Teamcenter Rich Client does not launch

If the **Send to Creo** button in the Teamcenter Rich Client does not launch Creo® and initiate the integration’s **Open** dialog, validate the following:

- The **ProE.bat** file exists in the **%TC_ROOT%\portal** directory and its contents point to the proper installation of the integration. The file is created during the installation of the integration client.
- After the **Send to Creo** button is selected, validate that **from_portal.txt** and (if sent from Structure Manager), **from_portal.xml** are created in the **%LOCALAPPDATA%\ipem** folder.
- If Creo® has started, but the open dialog does not appear, verify that the **iticreo_async_client.exe** process is running. This additional executable must be running whenever Creo® is started for all features of the integration to connect to Creo® asynchronously. Ensure that the executable is not blocked from running by virus protection or similar software.

Send to Creo® from Active Workspace does not work

If the **Send to Creo** button in the Teamcenter Active Workspace does not launch Creo® and initiate the integration’s **Open** dialog, validate the following:

- After the **Send to Creo** button is selected, validate that **from_aw.txt** is created in the **%LOCALAPPDATA%\ipem** folder.
- If Creo® has started, but the open dialog does not appear, verify that the **iticreo_async_client.exe** process is running. This additional executable must be running whenever Creo® is started for all features of the integration to connect to Creo® asynchronously. Ensure that the executable is not blocked from running by virus protection or similar software.

Renaming Models Imported from Pro/PDM®

Models that were once managed by Pro/PDM® may carry an internal lock that prevents Creo® from renaming them. The Integration will usually report only a general error when Creo® cannot rename a model, but if you were to use Creo®’s **File** | **Rename** command to rename such a model, you might see the following message:

Note

You may not rename an object that is managed by PDM from Creo®. Perform the rename from within PDM.

If you encounter this problem, you can override the lock by setting the Creo® configuration option `let_proe_rename_pdm_objects` in a `config.pro` file:

```
let_proe_rename_pdm_objects yes
```

This lock is not present in models imported from Pro/INTRALINK® unless they were also managed by Pro/PDM at one time.

BOM Synchronization

BOM synchronization can occur during Open from Teamcenter, Save to Teamcenter, or both:

- Open from Teamcenter
  - BOM synchronization is performed when the user, or the integration, has selected the **Update Assembly from BOM** option.
- Save to Teamcenter
  - BOM synchronization is automatic and unconditional.

The integration uses `ProFeatureID` occurrence notes to match features in the Creo® assembly with occurrences in the BOM. During Open, when the integration updates the assembly to match the BOM, it deletes components whose feature IDs cannot be matched to a `ProFeatureID` occurrence note. New occurrences created in Structure Manager do not need `ProFeatureID` notes if those components are not yet in the Creo® assembly, but occurrences of components already in the Creo® assembly must have a valid `ProFeatureID` note.

For example, suppose an assembly was migrated from Pro/Intralink to Teamcenter. The migration process does not create the `ProFeatureID` occurrence notes but does create the assembly product structure and all other MetaData. When you ask the integration to update the assembly to match its BOM, you may see that existing features are deleted from the assembly and replaced by new features for the same components, resulting in loss of assembly constraints. This occurs because the integration could not find `ProFeatureID` notes for those components. If the BOM does not have `ProFeatureID` notes for components that already exist in the Creo® assembly, first open the assembly in Creo® without updating it to match the BOM and save it back to Teamcenter. This will add `ProFeatureID` occurrence notes, after which you can modify the BOM in Structure Manager, and update the assembly, without losing assembly constraints for existing components.

Opening seed parts from Teamcenter

Teamcenter users can create items and datasets for new Creo® models in Teamcenter, and Creo® users can open the models. The integration can be configured to provide a template, or “seed” part, to populate the empty dataset at time of open. The user guide states that empty datasets used in this way must not have a revision extension, but this constraint is often overlooked because Teamcenter will automatically assign the revision extension as part of the dataset name. The user must remember to remove the extension before the seed part can be opened in Creo®. The integration displays this error when opening a seed part with the revision extension:

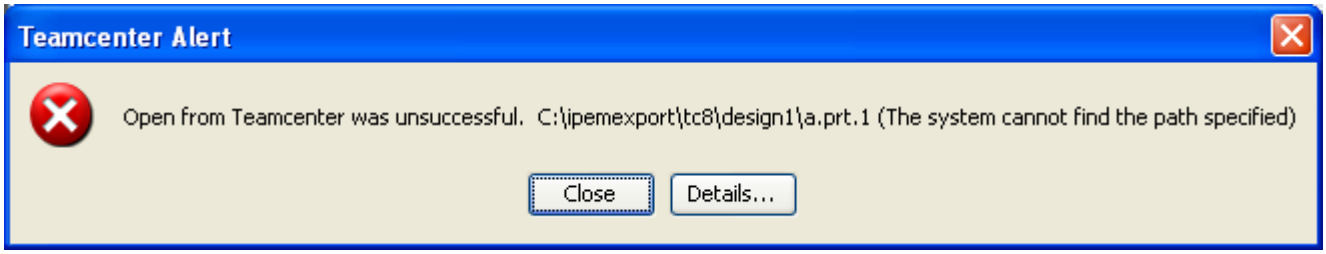


Figure 48 - Trying to open a dataset named with the item revision level

Opening pre Creo® 4.0 simplified representations from Teamcenter

Assemblies prior to Creo® 4.0 contained predefined simplified representations such as Geometry Rep and Graphics Rep. The integration stored an entry for each pre-defined rep into Teamcenter. pfcExceptions$XtookitBadInputs errors or java.lang.NullPointerException errors may occur while attempting to open such predefined simplified representations. Setting `hide_pre_creo4_reps=no` in Creo's config.pro file resolves the open problem, but also results in storing more predefined simplified representations into Teamcenter.

Teamcenter operations on Creo® data

Several Teamcenter operations, when executed on Creo® data, rely on code within the `libtxd` and `libtxdipem` shared libraries that are installed by the integration. Examples are:

- Revise
- Save As
- Rename
- Duplicate
- Baseline

When the `libtxd` and `libtxdipem` shared libraries cannot be found by the Teamcenter server process, or cannot be loaded for some reason, the operation will fail. An error message may be displayed, or the failure may be silent, and only discovered later. A typical example is revising of a Creo® assembly. Teamcenter creates the new item revision and copies the forms and datasets to the new revision. Teamcenter also creates the BOM View Revision (BVR) object for the new revision. But the Creo® integration is responsible for propagating the assembly's CAD dependency relations, and if the code within the `libtxd` and `libtxdipem` libraries cannot be executed, this does not happen. The result is an assembly revision without CAD dependencies, which leads to a regeneration failure in Creo®.

This problem can be identified by searching for the string "txd" in the Teamcenter system log file. If entries like this are found:

```
>>>> Error code  3600004 caught in file
"D:\tc810x_win\src\core\tccore\custom_itk.cxx" at line number 360

*** NOTE:: in line 362 of D:\tc810x_win\src\core\tccore\custom_itk.cxx at Thu Apr
14 15:54:38 2011 Eastern Daylight Time

+++ Could not load the Custom Library <libtxd> and/or Function Ptr
<libtxd_register_callbacks>
```

The Teamcenter Administrator should check the `TC_ROOT\bi`n directory and verify that the `libtxd` and `libtxdipem` file exists, and that it is the correct version of the file. This problem has been seen most often when the customer installs the integration on a 2-tier Teamcenter client workstation.

To resolve the problem on a 2-tier client, run the installer again. To resolve the problem in a 4-tier deployment, reinstall the integration's BMIDE template, using the Teamcenter Environment Manager (TEM) utility.

If the `libtxd` and `libtxdipem` libraries are loading successfully, but the Teamcenter operation is not working as expected, it is useful to set the **TXD_DEBUG** environment variable to a value of **true**, in the tcserver process's startup environment. This will cause the integration to write its own log messages to the tcserver system log, prefaced by the string **LIBTXD_DEBUG**. For example:

```
LIBTXD DEBUG - 17:45:31 - TXD_deepCopyAction - Start - Old Item Id:
MBD_CREO_PMI_COLOR, Old Revision Id: A, New Item Id: MBD_CREO_PMI_COLOR, New
Revision Id: B, Operation: Revise, copyied objs: 3
```

If the following error message occurs: A *"Creo dependency" relation was not created between "xyz" and "xyz" because it already exists*. The message is likely caused by a problem with the libtxd.dll (.so). The libtxd is installed on the server at %TC_ROOT%\bin\ directory. The file is shared between the Creo® and SOLIDWORKS® integrations to Teamcenter. Verify that the **Product Version** field of libtxd.dll (.so) is the same as the most recent release of the integrations. On Windows perform **RMB->Properties->Details->Product Version**. On Linux set TXD_DEBUG environment variable to "true" when pool manger is started, then the syslog will contain debug statements indicating the libtxdipem version.

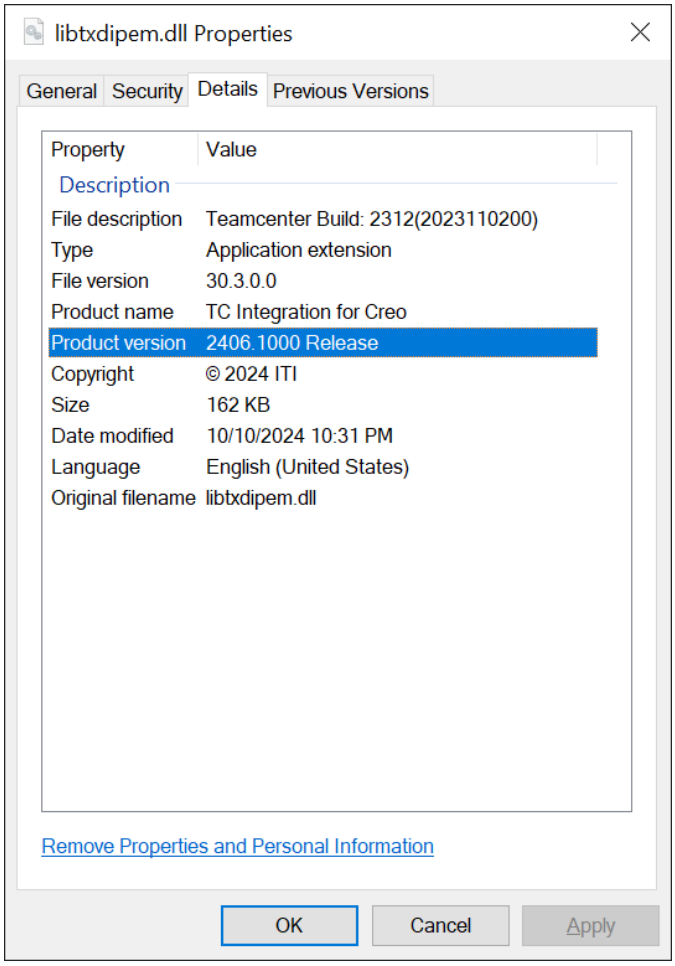


Figure 49 – For windows, select the DLL and click RMB -> Properties -> Properties: Details

Failure when saving or importing Creo® family tables

The integration invokes Creo's ModelCHECK utility when saving part and assembly family tables to Teamcenter, for optimal performance.

Depending upon the client's locale settings, the save or import operation may fail, with these errors:

```
INFO (  405.697): PERF: CREO call starts - PEDLLHelperImpl.runModelCheck

DEBUG (  405.775): OperationMonitor disabled

WARNING (  405.775): The integration has encountered an error.  Do you want
to complete the operation using slower processing?

WARNING (  405.807): The integration has encountered an error.  The likely
causes for the error are that MC_ENABLE not set to "Y" in config_init.mc in
the Creo installation or that the modelcheck_enabled option is set to "no"
for the current Creo session or that the <ipem install>\config directory is
modified or missing.
```

In addition to the suggested reasons, this can result from a mismatch between the client's runtime locale settings and the installed Creo® locales. Verify that the **%CREO_DIR%\Common Files\modchk\text** folder contains a subdirectory for the current client's operating system locale.

Creo membership relation not created

If a message such as 'A "Creo membership" relation was not created between "KSA5_20230826" and "KSC5_20230826/A;1", because it already exists..' occur when revising or performing a Structure Manager Duplicate, there is a problem with the libtxd library on the server. Most likely a template installation of an earlier version of the Teamcenter Integration for Creo or Teamcenter Integration for SOLIDWORKS was performed since the current version was installed. Try reinstalling the newer template to get the proper libtxd library installed on the server.

Failure to duplicate Creo® data

The Teamcenter Integration for Creo® has limited support for the Teamcenter Structure Manger Duplicate operation. The solution has three major components, and each must be understood in order to effectively troubleshoot problems.

Before proceeding, review the ReleaseNotes.pdf for **KNOWN ISSUES**.

Proper Installation

Three major components need to be installed correctly.

1. Integration server code – The server code runs when actions occur within the Teamcenter Duplicate dialog. There are two DLLs that must be installed in the `TC_ROOT\bi`n.
   a. `libtxd.dll`
   b. `libtxdipem.dll`
   c. `libPro2Ext.dll`
2. Dispatcher Translation Service – The ITI "creoduplicate" service is a translator that post-processes the resultant replica assembly structure from the Teamcenter Duplicate operation.
3. Dispatcher Client - See the InstallationGuideForDispatcherClient.pdf document.

Notification

The user is notified when the Dispatcher Client has completed the duplication of the Creo® data. The notification is sent to the user's Teamcenter Mailbox. The notification message indicates success or failure of the Duplicate operation. Until the user receives a notice of successful completion, the replicated models cannot be safely opened in Creo®.

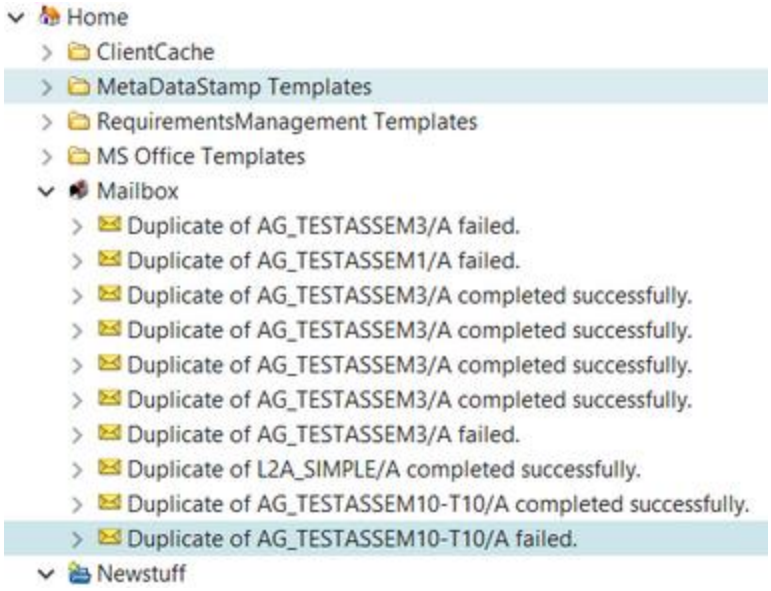


Figure 50 – Teamcenter notification of completed duplication

Common Problems

Here are some common problems, and tips on how to diagnose them.

1. The Teamcenter Duplicate dialog hangs, crashes or other. Was the integration server installed properly? Is it compliant with the Teamcenter versions? If yes to both, there may be a special data condition or problem in the dialog or integration code. Try removing the `libtxd.dll`, `libtxdipem.dll` and `libPro2Ext.dll` and repeating the duplicate. If the problem remains, then it is probably an issue with the Teamcenter Duplicate framework.
2. The datasets of the duplicated items are not renamed. This usually indicates that the server code is not installed properly. Make sure both libtxd.dll and llibPro2Ext.dll are present in the TC_ROOT\bin directory of the Dispatcher module server.
3. If users receive the following message while opening Creo® data:

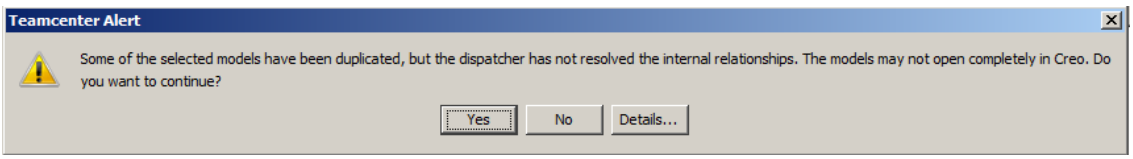


Figure 51 - Teamcenter Duplicate failure warning

   a. The creoduplicate service on the on the Dispatcher module server has not yet resolved the replica data.
      i. Use the Dispatcher Request Administration Console to determine the state of the request. To easily find requests, filter the requests by `Provider=ITI` and `service=creoduplicate` and `state=ALL`.

ii. If the request is in the INITIAL state, then the translation has not occurred yet, and the user either needs to wait, or the administrator must determine why the requests are not being processed.

iii. If the request is in the TERMINAL state, then it failed in execution. The task can be restarted. If it fails again, check the logs on the dispatcher server.

iv. A TERMINAL task can also be completed by command line execution of the `ipemduplicate.bat` program. Usage is self-documented via the –help option. To validate that the dataset is the top level of the duplicate operation, confirm the existence of a `Pro2_DuplicateInstructions` named reference within the dataset.

v. If the request is in the TERMINAL state and the `Pro2_DuplicateInstructions` named reference is not present, then a user must open the models into session and fix them manually. The `Pro2_DuplicateInstructions` named reference is removed by the dispatcher service when the parts and assemblies are resolved. It may be that 2D models such as drawings and layouts were not resolved when the dispatcher process went TERMINAL. These 2D models must be resolved manually by bypassing the dialog and reconciling the models in a Creo® session.

vi. If the request is in the TERMINAL state and make sure the user did not include the # character in the new Item ID of any BOMLines. The # character is not be resolved by the integration code.

Performance troubleshooting

Customers often report that saves and loads from and to a CAD integration client are slower than expected. This may be due to environment/infrastructure problems, or inefficiencies in the Teamcenter or integration code. In all cases, the recommended approach is to deploy Teamcenter performance journaling on both the Teamcenter enterprise tier and the CAD integration client. The resulting journal files provide a comprehensive view of where the performance bottlenecks exist and can help to guide Support Center and/or Product Development to the appropriate next steps.

Server performance journaling is documented by Teamcenter and is not covered here.

Deploy performance journaling on the integration client

In **ipem.properties**, set `log.pjl = true`. The resulting journal file will have the same name and location as the txdlog file.

Test case execution

1. Launch the integration, and perform the test case in as few steps as possible
2. Exit from the CAD session. This will automatically end the corresponding Teamcenter client session

3. Copy the client-side journal file to a safe location. This .pjl file will be located in the same directory as the txdlog.txt file and will have the same name except for the file name extension.

Chapter 8 Best Practice Recommendations

Properties display in the Save As, Save All, Save Checkouts and Teamcenter New Dialogs

The **Save** and **Teamcenter New** dialogs have two properties panels: **Required Properties** and **Optional Properties**. Initially, these panels contain only the properties required by OOTB Teamcenter. OOTB **Required Properties** are **Item Type**, **Item ID**, **Item Revision**, and **Item Name**. Both sets of properties may be distributed across the Item, master forms, and dataset types:

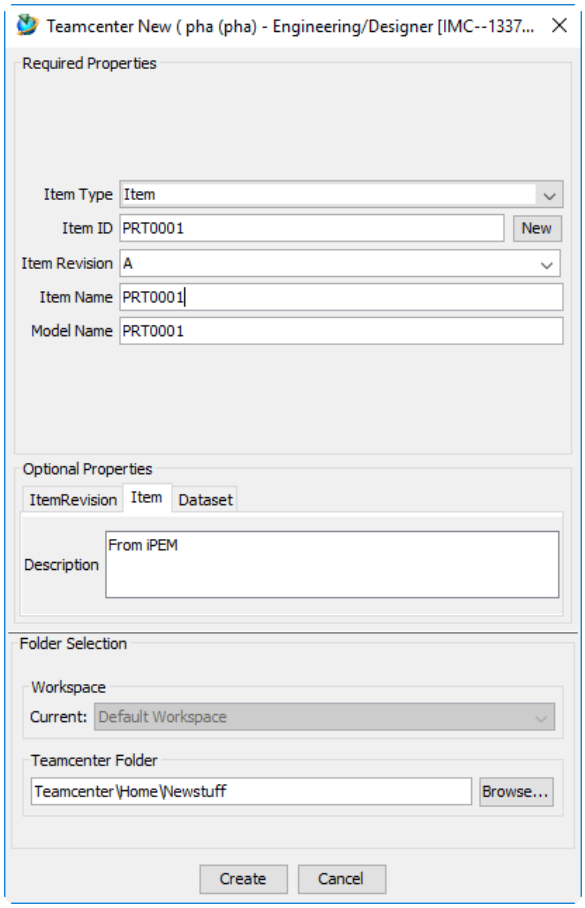


Figure 52 - OOTB Teamcenter Required and Optional properties

The visibility and read/write status of each property are controlled by Teamcenter *Operation Descriptors,* which are defined in both OOTB and custom BMIDE templates. The integration recognizes and enforces the ***Create, Revise*** and the ***SaveAs*** operation descriptors. Thus, the appearance and editable status of a given attribute may be different in the various dialogs, based on the context of the operation. Specifically:

Teamcenter New	The **Create** operation descriptor is enforced.
Save Operations	Either the **Create, Revise** or the **SaveAs** operation descriptor may be enforced. Because the **Save** dialog is used for multiple operations, the integration automatically applies the appropriate descriptor. If an item is being created for the first time, the **Create** operation descriptor is applied. If an item is being revised, the **Revise** operation descriptor is applied. If an existing item is being saved to a new item, the **SaveAs** descriptor is applied. If a new version of an existing item revision is being saved, no descriptor is applied, and no attributes may be assigned or modified. See Properties enabled in Save Dialog for Second Save for more information on the last operation.

In all **Save** dialogs, the ⊘ icon appears in the left most column when the model is selected for save, but one or more required properties has not been assigned a value.

Required and **Optional** properties may be included in attribute mappings. When an attribute mapping exists for a property, its value in the properties panel will initially be determined by the value of the mapped Creo® property. Mapped properties have the advantage of being visible to the user from within Creo®. When a user assigns a value to a mapped property in the properties panel, that value is written back into the Creo® file when the **Save** button is selected. Read more about attribute mapping in the Attribute Mapping section of this document.

Note

Create descriptors can differ between Item Types.

Models existing in Teamcenter without values for required properties can be saved without specifying a value.

Do not change the value of a mapped function-driven property in the **Save** dialog. The functional value will persist and user changes in the **Save** dialog will not persist.

Limit required create descriptors to properties which CAD designers know at time of first save. An excessive number of required properties might discourage designers from saving data to Teamcenter.

Configuring Teamcenter for Single Workflow

The integration can be configured to submit Item Revisions to a workflow process by adding

“workflow” to the `table.columns.checkin` preference in ipem.properties. The integration does not prevent users from submitting objects to multiple workflows. Sites that want to prevent this should set the Teamcenter preference **EPM_multiple_process_targets** to “OFF”. Teamcenter 8.3 and beyond set **EPM_multiple_process_targets** = ON, by default.

Restricted access to data

While opening CAD data from Teamcenter, and while saving CAD data to Teamcenter, the integration identifies all models that are related to the top-level structured model (drawing, assembly, family table generic). During **Open** and **Save**, Teamcenter relationships are expanded and traversed. During **Save**, the CAD session is also interrogated to read its network of dependency relations.

While expanding and traversing these relations, Teamcenter may encounter one or more components that are inaccessible in one way or another. Common scenarios are:

- The item, item revision, dataset, or some combination of these, has been assigned to a project to which the CAD user does not have read access.
- The item, item revision, dataset, or some combination thereof, has an ACL attached, or is owned by a security-enabled group, that explicitly or implicitly denies read access to the CAD user.
- Any of the above objects was imported into the local database via multi-site import, and has since been deleted, leaving a stubbed reference in the local Teamcenter database. Teamcenter allows deletion of replica data even though it is referenced by higher-level objects in the local database. When that replica data is deleted, the reference remains.
- The user is saving an assembly that has suppressed components, and those suppressed components have never existed in Teamcenter. This can occur when the components remain suppressed the first time the assembly is saved to Teamcenter. Because they are not available in session, the integration cannot save them to Teamcenter as items[7].

Because inaccessible models may result from deliberate customer security practices, such as assignment to secure projects, the integration does not consider them an error condition. Instead, it informs the user about the situation during both Open and Save operations.

Opening a model with access restrictions

When a model with restricted components is opened from Teamcenter, the integration displays an informational dialog, stating how many inaccessible dependencies were encountered. Depending upon the user's access to the objects involved in the operation, additional details such as Item ID and model name may also appear in the dialog:

[7] This situation can be prevented by setting `checkin.ignoremissing = never` in ipem.properties. This is not the default setting, because it is very common for customer data to contain "ghost references", which are pointers to non-existent, non-required dependencies. With the default setting of "prompt", the integration warns the user about these missing references, but does not prevent him from completing the save.

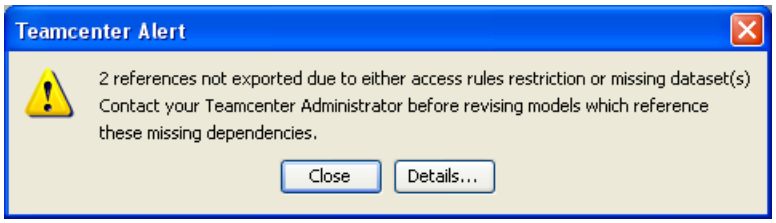


Figure 53 - Opening an assembly with restricted components

Creo® Skeleton Models

The integration save and import exclude Creo® Skeleton models from the Teamcenter BOM. A Creo Skeleton relation is created from the assembly dataset to the skeletons Item Revision.

Sites should establish a practice to either store skeletons in their own Item or in the Item of the assembly they are used in. If a skeleton is used in multiple assemblies, it must be stored in it's own Item. If a skeleton is only used by a single assembly, it can be stored with that assembly (Note: In general, parts are not allowed to be stored in an Item with an assembly, but skeletons are an exception). If any skeletons could be used by multiple assemblies, then it is best to store the skeletons in their own Item. The best practice is to always store skeletons in their own Item.

MutliField Keys

The integration does not support Teamcenter Reference type properties in the definition of a MultiField Keys forbusiness object. Use string type instead.

Format Files

The recommended practice for managing and working with Creo® Format files (.frm) is as follows.

- Administrator checks in CAD .frm files which exist in the Creo "pro_format_dir" location.
- Administrator Releases the .frm Items in Teamcenter to prevent unauthorized overwrite.
- Administrator checks out these .frm files to the creo "pro_format_dir" location so that the files are available to the user.
- The ipem.txr file should be retained in the "pro_format_dir". This should prevent update or new hints from appearing on the IPEM dialogs for the format files.
- Administrator is in charge of keeping format files up to date. Users follow Creo procedures when creating drawings, selecting formatsfrom Creo®location

Chapter 9 Java Customization points

It is possible to implement custom Java code that the integration will execute when specific actions occur within the integration. The code must be compiled into an IpemUserExits.class file and placed in the `<ipem_dir>\com\transcendata\userexits folder`. When the file exists in that location, the integration will import and execute the implemented user exit methods. It is not required that all user exits be implemented - any number of the user exits can be implemented from within the class file.

Start with the `<ipem_dir>\com\transcendata\userexits\IpemUserExits.txt` file. It has an extensive sample implementation. The comments within the file explain how to compile it, and the sample code indicates how to use each user exit, process the XML, and formulate the proper return and response. User exits are provided to augment the functionality provided by the integration. Customers can implement code to validate user actions or customize actions performed by the integration.

Note

Always compile the `IpemUserExits.java` file with a Java JDK version that is compatible with the runtime java version installed on the clients.

Available User Exits

saveInit()	This method is invoked before the **Save** or **Teamcenter New** Dialogs are displayed. The integration sends an XML file containing the properties to be displayed in the **Save** dialog. The customization returns an XML file listing new models and the property values to pre-populate when the dialog is displayed.
saveItemId()	This method is invoked when the user enters a value in the **Item ID** button in the **Save** and **Teamcenter New** dialogs. The integration sends an XML file containing the data in the save dialog and an indication of which models were selected. The customization returns an XML file listing models to be renamed, and the new Item IDs and Model Names to be assigned.
saveNew()	This method is invoked when the user clicks on the **New** button in the **Save** or **Teamcenter New** Dialogs. The integration sends an XML file containing the data in the save dialog and an indication of which models were selected. The customization returns an XML file listing models to be renamed, and the new Item IDs and Model Names to be assigned.
saveReplace()	This method is invoked when the user clicks on the **Replace** button in the **Save** or **Teamcenter New** Dialogs. The Integration's own **Replace** dialog does not appear. The integration provides an XML file containing the data in the save dialog, and an indication of which models were selected for

	Replace. The customization returns an XML file listing models to be renamed, and the new Item IDs and Model Names to be assigned.
saveValidation()	This method is invoked when the user selects the **Save** button on the **Save** Dialog or **Create** button in **Teamcenter New** Dialog. The integration writes the relevant information from the save dialog to an XML file for the customer code to validate. The return is a Boolean and a value of true indicates the user exit validation passed.

Sample code is provided in `<IPEM_DIR>\com\transcendata\userexits\IpemUserExits.txt`. The file should be renamed to `IpemUserExits.java` and compiled within the same directory. The sample code demonstrates how to access the information in the XML files.

Integration generated XML

The integration generates an XML file before calling each user exit. The XML file contains the information required for the user exit. For the save user exits, the content comes directly from the **Save or Teamcenter New** Dialogs. The XML file is generated in the user's `%TEMP%` directory. The structure of the XML file is the same for each user exit. The following is a description of the tags:

Tags

ipem_user_exit

Root tag for the action being performed. Only one `ipem_user_exit` tag will exist in the XML file. All other tags are nested in the `ipem_user_exit` tag. The following is a list of the attributes of the `ipem_user_exit` tag.

`action`	This string specifies the action performed by the user, which corresponds action occurring. Possible values are one of **`init`**, **`item id`**, **`new`**, **`replace`**, or **`validate`**
`after_checkin_action`	This string indicates which radio button has been selected in the **Action on Save** panel of the **Save** dialog. Possible values are Delete **`Files`**, **`Retain Files`**, **`Retain Files and Checkouts.`**
`group`	This string contains the group to which the current user's session is assigned.
`item_id_value`	This string contains the text entered in the Item ID field. This attribute is only valid for the **saveItemId()** user exit.
`item_id_value_existing`	A `yes` or `no` string which indicates if the Item specified by `item_id_value` is existing in Teamcenter. A value of `yes` indicates the Item already exists in Teamcenter. This attribute is only valid for the **saveItemId()** user exit.
`operation`	This string specifies the operation being performed by the user. Operation values include: CHECKIN_MODIFIED, CHECKIN_AS, CHECKIN_ALL_MODIFIED, CHECKIN_ALL_AS, RMB->SAVE_AS, RMB->SAVE_AS_BELOW, REGISTER_NEW, UTILITY_IMPORT.

`project`	This string specifies the project to which the current user's session is set.
`role`	This string specifies the current user's role in his group.
`user_id`	This string contains the current' user's Teamcenter login ID.

auxiliary_files

This is an optional sub tag of the `ipem_user_exit` tag. It does not have any attributes. It contains one or more `cadtopdm_control` tags.

cadtopdm_control

Each `auxiliary_files` tag contains one or more `cadtopdm_control` tags. This tag indicates the auxiliary files checkbox settings.

`label`	This string contains the same text that appears next to the auxiliary file check boxes on the **Save** dialog. For example, **Save Creo® Models** or **Save JT Files**.
`value`	A `yes` or `no` string which indicates whether the check box was enabled when the user exit was called from the **Save** dialog.

file_name

Each `model` tag contains a `file_name` tag.

`value`	This string contains the model's file name, including its file name extension. For example, **`part1.prt`**. For a family table instance, the value of the **`file_name`** tag is that of the instance's master generic.

item_id

Each `model` tag contains an `item_id` tag.

`value`	This string contains the model's currently assigned Item ID.

item_name

Each `model` tag contains an `item_name` tag.

`value`	This string contains the mode's currently assigned Item Name

item_revision_id

Each `model` tag contains an `item_revision_id` tag.

`value`	This string contains the model's Item Revision ID.

item_type

Each `model` tag contains an `item_type` tag.

`value`	This string contains the model's current Item Type.

model

A `model` tag is created for each row in the table of models. The following attributes are included in the `model` tag:

`first_save`	Possible values are **yes** and **no**. A value of **yes** indicates this is the first time this model is being saved to Teamcenter.
`master`	A string containing the name of the master generic of an instance. If the model is not an instance, then **master** will be an empty string.
`name`	The name of the model. For a family table instance, this is the instance name.
`save`	Possible values are **yes** and **no**. A value of **yes** indicates that the model has a check in the save column.
`selected`	Possible values are **yes** and **no**. A value of **yes** indicates that the model's row in the table was selected (highlighted) when the action was performed. This attribute is not relevant for the saveInit() or saveValidation() user exits.
`type`	This string indicates the model's type. Possible values are those included in the `dataset_map` section of the ipem.xml file.

properties

A `model` tag contains a `properties` tag, which in turn contains `pdm_property` tags for those model attributes with Operation Descriptors in the Teamcenter schema definition. Each `pdm_property` tag contains one or more `value` tags for the property.

pdm_property

The following attributes are included in the `pdm_property` tag:

`name`	The name of the property.
`required`	Possible values are **yes** and **no**. Yes indicates the property is required to complete the save operation.

value

There are one or more value sub tags for each `pdm_property` tag. The `value` tag only contains a value, it does not have any attributes. If the property is an array, multiple `value` tags can be present, and each will have an `index` attribute such as `index="0"`.

Sample XML generated by the Integration

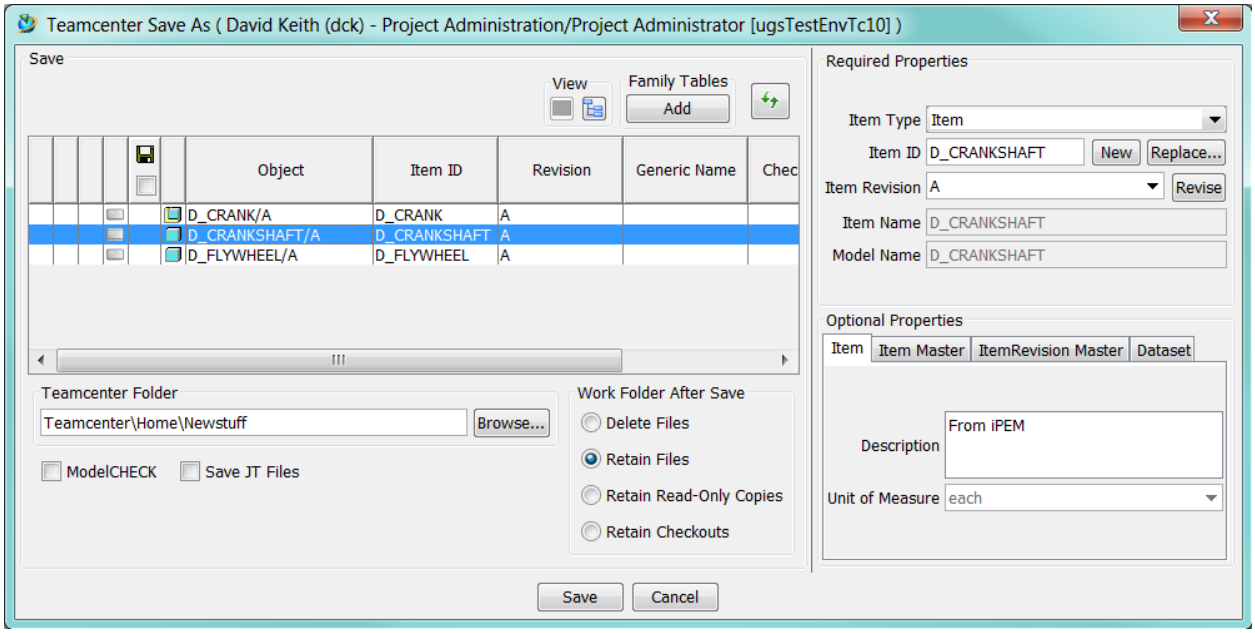


Figure 54 The replace of a simple assembly referencing D_CRANKSHAFT that is given a new Item ID of BLUE_CRANKSHAFT.

This **Save** dialog contains an assembly `D_CRANK` that references three components `D_CRANKSHAFT` and `D_FLYWHEEL`. When the user presses the **Replace** button with the `D_CRANKSHAFT` selected, the following XML is generated:

```
<?xml version="1.0" encoding="UTF-8"?>
<ipem_user_exit action="replace"
after_checkin_action="Retain Files" group="Project
Administration" operation="Save As" project=""
role="Project Administrator" user_id="dck">
    <auxiliary_files>
        <cadtopdm_control label="Save Creo Models"
value="yes"/>
        <cadtopdm_control label="Save JT Files"
value="no"/>
    </auxiliary_files>
    <model first_save="no" name="D_CRANKSHAFT" save="no"
selected="yes" type="PRT">
        <item_type enabled="yes" mfk="no" value="Item"/>
        <item_id enabled="yes" mfk="yes"
value="D_CRANKSHAFT"/>
        <item_revision_id value="A"/>
        <item_name enabled="no" mfk="no"
value="D_CRANKSHAFT"/>
        <file_name value="d_crankshaft.prt.5"/>
        <child_models>
            <child name="D_DRILL" type="ASM"/>
        </child_models>
        <properties>
            <pdm_property enabled="no" mfk="no"
name="Item.uom_tag" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="yes" mfk="no"
name="Item.object_desc" required="no">
                <value>From iPEM</value>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.object_desc" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.item_comment" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.previous_item_id" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.project_id" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.serial_number" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.user_data_1" required="no">
                <value/>
            </pdm_property>
```

```
            <pdm_property enabled="no" mfk="no" name="Item
Master.user_data_2" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.user_data_3" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.object_desc" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.item_comment" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.previous_version_id"
required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.project_id" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.serial_number" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.user_data_1" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.user_data_2" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.user_data_3" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="yes" mfk="no"
name="Dataset.object_desc" required="no">
                <value>From iPEM</value>
            </pdm_property>
        </properties>
    </model>
    <model first_save="no" name="D_FLYWHEEL" save="no"
selected="no" type="PRT">
        <item_type enabled="yes" mfk="no" value="Item"/>
        <item_id enabled="yes" mfk="yes"
value="D_FLYWHEEL"/>
        <item_revision_id value="A"/>
        <item_name enabled="no" mfk="no"
value="D_FLYWHEEL"/>
        <file_name value="d_flywheel.prt.5"/>
        <properties>
```

```
            <pdm_property enabled="no" mfk="no"
name="Item.uom_tag" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="yes" mfk="no"
name="Item.object_desc" required="no">
                <value>From iPEM</value>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.object_desc" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.item_comment" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.previous_item_id" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.project_id" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.serial_number" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.user_data_1" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.user_data_2" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.user_data_3" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.object_desc" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.item_comment" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.previous_version_id"
required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.project_id" required="no">
                <value/>
            </pdm_property>
```

```
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.serial_number" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.user_data_1" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.user_data_2" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no"
name="ItemRevision Master.user_data_3" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="yes" mfk="no"
name="Dataset.object_desc" required="no">
                <value>From iPEM</value>
            </pdm_property>
        </properties>
    </model>
    <model first_save="no" name="D_CRANK" save="no"
selected="no" type="ASM">
        <item_type enabled="yes" mfk="no" value="Item"/>
        <item_id enabled="yes" mfk="yes" value="D_CRANK"/>
        <item_revision_id value="A"/>
        <item_name enabled="no" mfk="no" value="D_CRANK"/>
        <file_name value="d_crank.asm.6"/>
        <child_models>
            <child name="D_CRANKSHAFT" type="PRT"/>
            <child name="D_FLYWHEEL" type="PRT"/>
        </child_models>
        <properties>
            <pdm_property enabled="no" mfk="no"
name="Item.uom_tag" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="yes" mfk="no"
name="Item.object_desc" required="no">
                <value>From iPEM</value>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.object_desc" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.item_comment" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.previous_item_id" required="no">
                <value/>
            </pdm_property>
            <pdm_property enabled="no" mfk="no" name="Item
Master.project_id" required="no">
                <value/>
```

```
                </pdm_property>
                <pdm_property enabled="no" mfk="no" name="Item
            Master.serial_number" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no" name="Item
            Master.user_data_1" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no" name="Item
            Master.user_data_2" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no" name="Item
            Master.user_data_3" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.object_desc" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.item_comment" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.previous_version_id"
            required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.project_id" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.serial_number" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.user_data_1" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.user_data_2" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="no" mfk="no"
            name="ItemRevision Master.user_data_3" required="no">
                    <value/>
                </pdm_property>
                <pdm_property enabled="yes" mfk="no"
            name="Dataset.object_desc" required="no">
                    <value>From iPEM</value>
                </pdm_property>
            </properties>
        </model>
</ipem_user_exit>
```

Response XML from custom code

The **saveInit()**, **saveItemId(), saveNew()** and **saveReplace()** user exits must provide a response in the form of XML file written to the user's `%TEMP%` directory. This response is read by the integration and applied to the models in the **Save** dialog. Both user exits can generate new Item IDs and can rename the models in Creo®. The XML definition requires a new Item ID, and optionally a new name, for the models. The models need not have been selected, but they must have appeared in a model tag in the Integration-generated XML.

Response Tags

save_operation_response

This is the root tag for the response. Only one `save_operation_response` tag will exist in the XML file. All other tags are nested in the `save_operation_response` tag. The following are the attributes of the `save_operation_response` tag.

action

This string specifies the action being performed within the operation. Possible values are **new**, **replace**, and **validate**.

model

The `save_operation_response` contains one or more `model` tags. A `model` tag is created for each model to be modified from the response XML. The `model` tag's `master`, `name`, and `type` attributes must match those of a model tag in the Integration-generated XML. The `new_item_id` and `new_model_name` attributes contain the modified values to apply to corresponding rows in the **Save** dialog. The following attributes are included in the tag:

`master`	A string containing the name of the master generic of an instance. If the model is not an instance, then **master** will be an empty string.
`name`	The name of the model to be processed. For an instance, this is the instance name.
`new_item_id`	The value must be that of a new Item Id. This is a required attribute. It is not permitted to change the model name only. The integration will not attempt to determine whether the Item ID already exists. The user exit must return a valid Item ID for the Item type.
`new_model_name`	The new name for the model (optional)
`type`	The type of the model, such as **ProPrt**, **ProDrw**, or **ProAsm**.

properties

A `model` tag contains a `properties` tag, which in turn contains pdm_`property` tags for those model attributes with Operation Descriptors in the Teamcenter schema definition. All properties provided in

the integration generated XML must also be in the response XML file. The `pdm_property` tag is the means to change Multi-Field properties. Item Type can be a Multi-Field key, but it cannot be changed through one of the user exits.

pdm_property

The following attributes are included in the `pdm_property` tag:

`name`	The name of the property as provided in the integration generated XML file. Additionally, Item Name and the Item Revision ID can be changed by adding pdm_property tag with `name="Item.object.name"` for Item Name and tag with `name="ItemRevision.item_revision_id"` for Item Revision Id.
`value`	This attribute contains the value of the property, as it is to be displayed in the **Save** dialog. If the value is empty, the property will be cleared. If the property is an array, multiple `value` tags can be present, and each will have an `index` attribute such as `index="0"`. It is the responsibility of the custom code to ensure a value is valid. When invalid values are included in a response, the results are unpredictable (an empty value may result, an alert may result, a message might be logged, etc…).

Sample Response XML

The following response XML will replace the `D_CRANKSHAFT` model with `BLUE_D_CRANKSHAFT`:

```
<?xml version="1.0" encoding="UTF-8"?>
<save_operation_response action="replace">
    <model master="" name="D_CRANKSHAFT" new_item_id="BLUE_D_CRANKSHAFT"
type="PRT">
        <properties>
            <pdm_property name="Item.uom_tag">
                <value/>
            </pdm_property>
            <pdm_property name="Item.object_desc">
                <value>From iPEM</value>
            </pdm_property>
            <pdm_property name="Item Master.object_desc">
                <value/>
            </pdm_property>
            <pdm_property name="Item Master.item_comment">
                <value/>
            </pdm_property>
            <pdm_property name="Item Master.previous_item_id">
                <value/>
            </pdm_property>
            <pdm_property name="Item Master.project_id">
                <value/>
            </pdm_property>
            <pdm_property name="Item Master.serial_number">
                <value/>
            </pdm_property>
            <pdm_property name="Item Master.user_data_1">
                <value>Replace Master 33</value>
            </pdm_property>
```

```
            <pdm_property name="Item Master.user_data_2">
                <value/>
            </pdm_property>
            <pdm_property name="Item Master.user_data_3"/>
            <pdm_property name="ItemRevision Master.object_desc">
                <value/>
            </pdm_property>
            <pdm_property name="ItemRevision Master.item_comment">
                <value/>
            </pdm_property>
            <pdm_property name="ItemRevision Master.previous_version_id">
                <value/>
            </pdm_property>
            <pdm_property name="ItemRevision Master.project_id">
                <value/>
            </pdm_property>
            <pdm_property name="ItemRevision Master.serial_number">
                <value/>
            </pdm_property>
            <pdm_property name="ItemRevision Master.user_data_1">
                <value>Replace Master 33</value>
            </pdm_property>
            <pdm_property name="ItemRevision Master.user_data_2">
                <value/>
            </pdm_property>
            <pdm_property name="ItemRevision Master.user_data_3">
                <value/>
            </pdm_property>
            <pdm_property name="Dataset.object_desc">
                <value>From iPEM</value>
            </pdm_property>
        </properties>
    </model>
</save_operation_response>
```

Chapter 10 VC++ Customization points

The integration loads managed C++ DLL libraries from the IpemUserExit.dll file located in `{IPEM_DIR}\bin`, then searches that library for types containing a method that matches an expected method signature. If XML for a user exit is present in `{IPEM_DIR}\ipem.xml` and a method is found that matches the signature, it's called by the integration during an operation.

Available User Exits

Below are the supported User Exits.

Method Visibility	Method Return Type	Method Name	Example File Types Provided	Supported Item Types
public	bool	AuxiliaryFileUserExit()	IGS (.igs)	Parts & Assemblies
			3D PDF (.pdf)	Parts & Assemblies
			3D STEP (.stp)	Parts & Assemblies
			Binary STL (.stl)	Parts & Assemblies
			TIF (.tif)	Drawings

AuxiliaryFileUserExit()

The AuxiliaryFileUserExit() method is called during Save operations. With the proper XML tags (see xml examples below), additional file types are created when performing a Teamcenter Save operation or from Dispatcher.

The AuxiliaryFileUserExit() uses the PTC Creo® C++ Object Toolkit API to interact with the session and perform the actions require to achieve the desired result.

Note

Only add Creo® API calls to the AuxiliaryFileUserExit() that are required. Using the API's to manipulate the session in a way the causes the Save operation to malfunction is not supported.

Example XML

These are examples of xml mappings that invoke the AuxiliaryFileUserExit capabilities within the Teamcenter Save dialog. The XML is placed in `{IPEM_DIR}\ipem.xml`. Complete documentation of the auxiliary_file mapping is in the Auxiliary Files section of this document.

Exported File Types	Example XML
IGS (.igs)	<auxiliary_file cad_type="ASM:PRT" direction="cadtopdm"> <pdm_location named_ref="MISC_TEXT" relation_type="IMAN_Rendering" pdm_type="MISC"/> <file_name pattern="{cad_name,lower}.igs"/> <cadtopdm_control label="Do IGES User Exit" user_preference_default="false"/> <user_exit file_name="{cad_name,lower}.igs" ignore_status="true"/> </auxiliary_file>
3D PDF (.pdf)	<auxiliary_file cad_type="ASM:PRT" direction="cadtopdm"> <pdm_location named_ref="PDF_Reference" relation_type="IMAN_Rendering" pdm_type="PDF"/> <file_name pattern="{cad_name,lower}.pdf"/> <cadtopdm_control label="Do 3D PDF User Exit" user_preference_default="false"/> <user_exit file_name="{cad_name,lower}.pdf" ignore_status="true"/> </auxiliary_file>
3D STEP (.stp)	<auxiliary_file cad_type="ASM:PRT" direction="cadtopdm"> <pdm_location named_ref="MISC_TEXT" relation_type="IMAN_Rendering" pdm_type="MISC"/> <file_name pattern="{cad_name,lower}.stp"/> <cadtopdm_control label="Do STEP User Exit" user_preference_default="false"/> <user_exit file_name="{cad_name,lower}.stp" ignore_status="true"/> </auxiliary_file>
Binary STL (.stl)	<auxiliary_file cad_type="ASM:PRT" direction="cadtopdm"> <pdm_location named_ref="STL-bin" relation_type="IMAN_Rendering" pdm_type="STL"/> <file_name pattern="{cad_name,lower}.stl"/> <cadtopdm_control label="Do STL User Exit" user_preference_default="false"/> <user_exit file_name="{cad_name,lower}.stl" ignore_status="true"/> </auxiliary_file>
TIF (.tif)	<auxiliary_file cad_type="DRW" direction="cadtopdm">

	<pdm_location named_ref="TIF_Reference" relation_type="IMAN_Rendering" pdm_type="TIF"/> <file_name pattern="{cad_name,lower}.tif"/> <cadtopdm_control label="Do TIF User Exit" user_preference_default="false"/> <user_exit file_name="{cad_name,lower}.tif" ignore_status="true"/> </auxiliary_file>

Modifying the User Exit DLL

Before opening Visual Studio project, set two required environment variables: **ITI_CREO_COMPILER_FLAG** & **ITI_CREO_DIR**. ITI_CREO_COMPILER_FLAG should be set to Creo® version of the integration (formatted as __CREO<version number>). **ITI_CREO_DIR** is the directory path should be set to the PTC Creo® directory. See table below for examples:

Environment Variables	Values
ITI_CREO_COMPILER_FLAG	`__CREO7`
	`__CREO8`
	`__CREO9`
	`__CREO10`
ITI_CREO_DIR	e.g., C:\Program Files\PTC\Creo 8.0.5.0

To modify the existing IpemUserExit.dll, start by opening the `{IPEM_DIR}\lib\ipemuserexits_cpp\IpemUserExit.vcxproj` into Visual Studio.

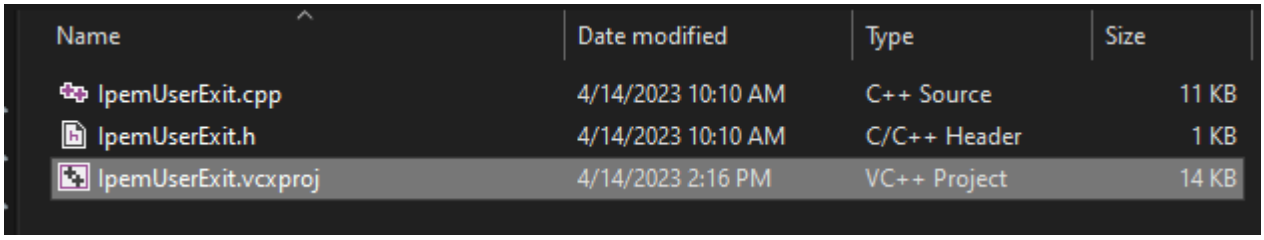


Figure 55 Open Project

Use the Solution Explorer to open both IpemUserExit.cpp and IpemUserExit.h as this is where modification will be made. In the top ribbon bar, ensure that the solution platform is x64 (and not x86). Also ensure that the solution configuration is **Debug** (and not release).

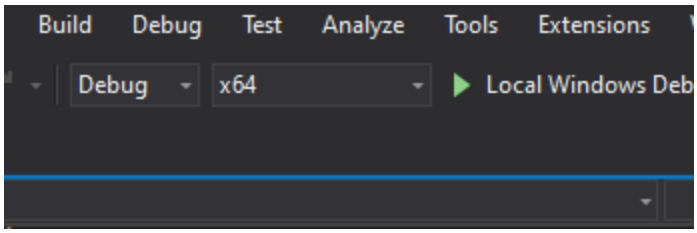


Figure 56 Verify solution platform/configuration

Start to make modifications to the source files as desired.

Note

Creo® Object Toolkit programming is beyond the scope of this document; please refer to the appropriate PTC resources.

Building and Installing Production IpemUserExit.dll

Once modification is complete, change the solution configuration from **Debug** to **Release**.

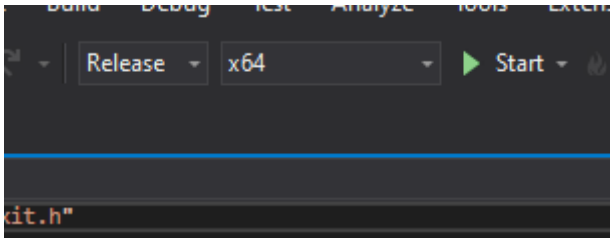


Figure 57 Change to Release

Build the solution by going to the top bar and selecting **Build > Build Solution**. Verify the solution built correctly.

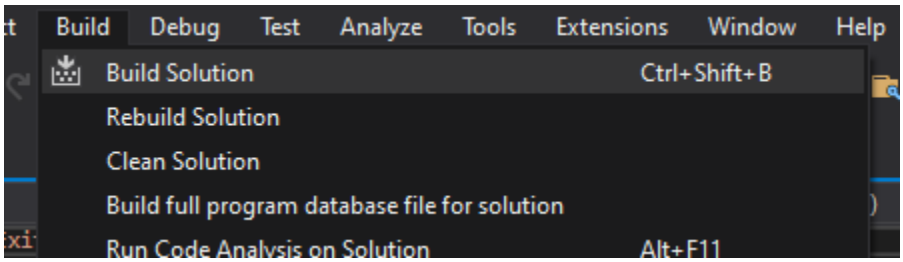


Figure 58 Build the Solution

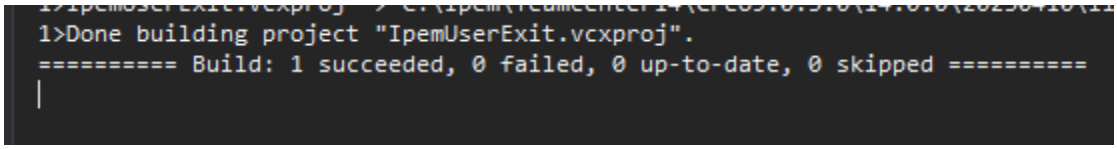


Figure 59 Verify the build succeeded

Copy the newly created `IpemUserExit.dll` file from the build folder and overwrite the existing IpemUserExit.dll in the bin folder.

From

`{IPEM_DIR}\lib\ipemuserexits_cpp\build`

To

`{IPEM_DIR}\bin`

Launch the Teamcenter Integration for Creo® to test the new `IpemUserExit.dll`

The Teamcenter Save Dialog should include checkboxes of the types involving the `IpemUserExit.dll` that are added in the xml file (`{IPEM_DIR}\ipem.xml`)

Check the boxes for the user exits the user exits.

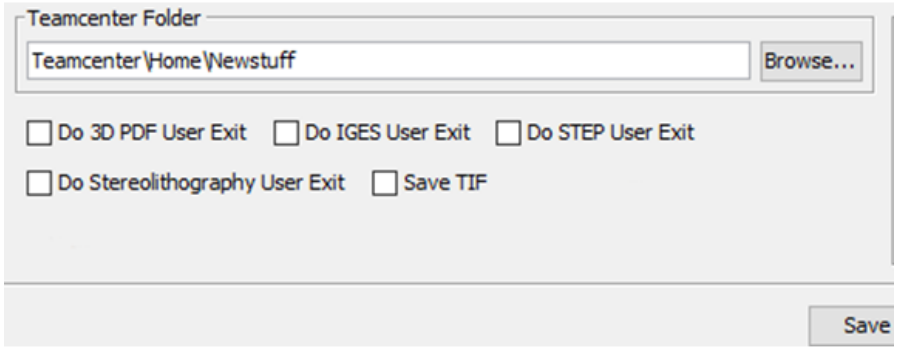


Figure 60 Checkboxes for the User Exit on the Save Dialog

Debugging IpemUserExit.dll

To begin debugging, ensure the solution platform is set to **Debug**. Build the solution.

Copy both the IpemUserExit.dll and the IpemUserExit.pdb from the build folder and overwrite the existing IpemUserExit.dll in the bin folder.

From

`{IPEM_DIR}\lib\ipemuserexits_cpp\build`

To

```
{IPEM_DIR}\bin
```

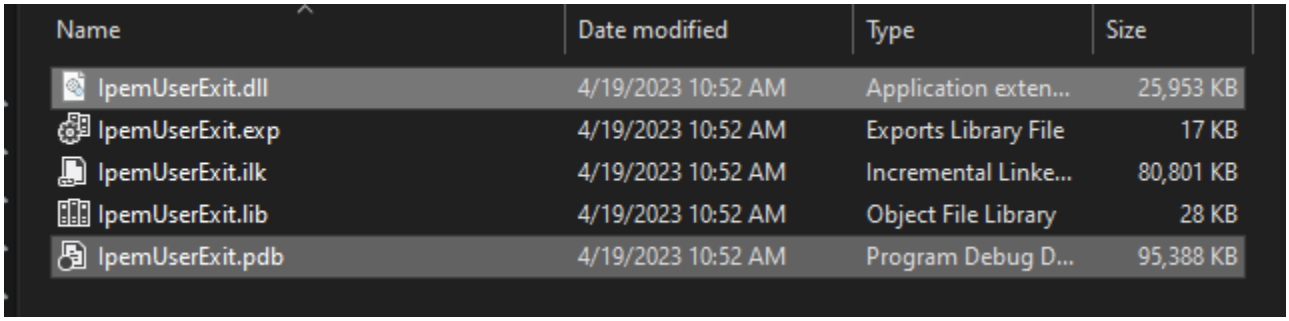


Figure 61 **Files for debugging**

After launching the integration, attach Visual Studio to the xtop.exe process. In Visual Studio, navigate to **Debug > Attach to Process**:

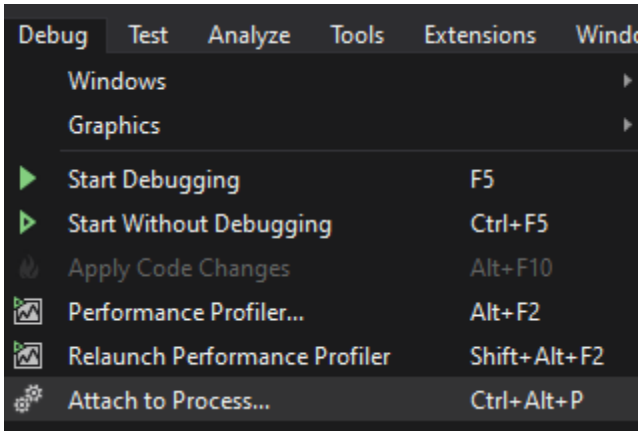


Figure 62 Attach to process.

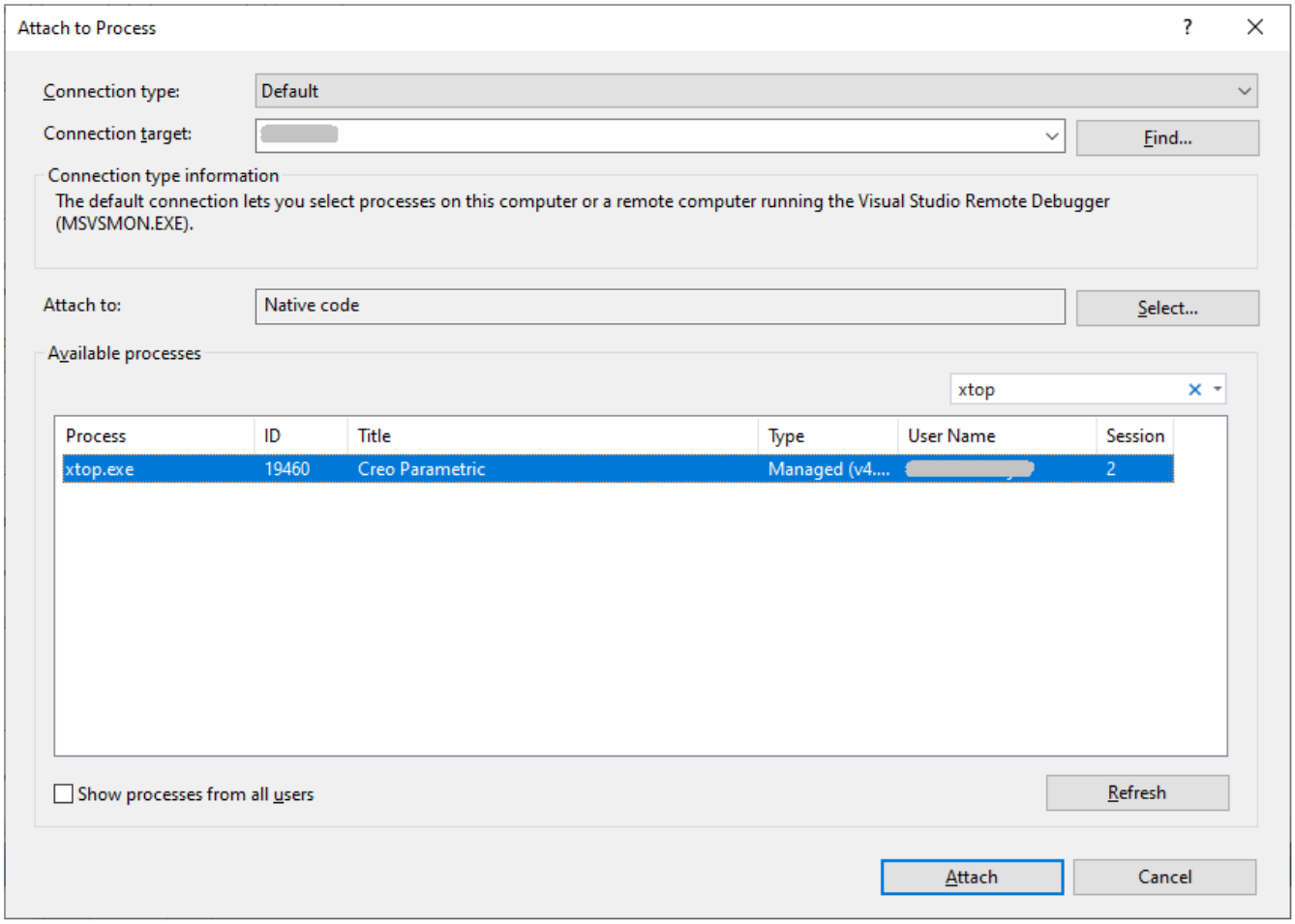


Figure 63 Attaching to Process

Make sure **Attach to:** is set to **Native code**

For example, if debugging AuxiliaryFileUserExit() method

Wait for the Creo® window to load before attaching to the process.

Place a break point in the required method

Run the integration operation and the breakpoint should be hit after the **Save** button is selected on the Teamcenter Save Dialog. assuming one user exit specific checkbox is selected.

Chapter 11 Appendix

Auxiliary files

This section describes the full range of options available to customers, for configuring auxiliary file mapping to and from Teamcenter. Many of these options are seldom used, and so they have been moved to the Appendix for clarity.

Specialized Auxiliary Files

JT Files

The next example shows all of the elements that may appear in a jt_file tag:

```
<auxiliary_file_map>
  <jt_file cad_type="ASM:PRT" action="translate" save="true">
    <translate_script template="{ipem_dir}\pejt\proetojt.bat"
      cmd="{eai_translate_script}"/>
    <create_script template="" cmd=""/>
    <configuration_file template="{ipem_dir}\pejt\proetojt.config"
      assembly_template=""/>
    <command_file template="{ipem_dir}\pejt\proetojt.txt"/>
    <file_name pattern="{cad_name,lower}.jt"/>
  </jt_file>
</auxiliary_file_map>
```

The `translate_script` tag indicates where to find a template for a script that generates JT files when the action is `"translate"` and the `create_script` tag indicates where to find an additional script template when the action is `"create_scripts"`. The `translate_script` and `create_script` tags also define an operating system command to execute each script. The `configuration_file` tag indicates where to find templates for EAI translator configuration files, and the `command_file` tag indicates where to find a template for the EAI translator command file. These tags are described in more detail below.

Note

The presence of a *`jt_file`* tag in the auxiliary file map makes the **Save JT Files** checkbox visible in the **Save** dialog. When this checkbox is on, the *`jt_file`* tag is enabled, and the integration generates

and/or saves JT files as directed by the *action* and *save* attributes. When this checkbox is off, the *jt_file* tag is disabled as if *action* is *"none"* and *save* is *"false"*.

The default state of this checkbox can be set with the *eai.jtenabled* user preference. This option and others are described in detail in the ipem.properties file, found in the directory where the Teamcenter Integration for Creo® is installed.

jt_file

```
<jt_file cad_type="types" [action="translate"|"create_scripts"|"none"]
[save="true"|"false"] [eai_dir="path"]> optional elements </jt_file>
```

The jt_file tag specifies the types of models for which JT files should be generated and/or saved to Teamcenter. The value for *types* can be either "PRT", if only part JT files are of interest, or "PRT:ASM" if part and assembly JT files should be generated and saved.

The action attribute indicates whether the Integration should generate JT files. The default action is "translate", which causes the EAI batch translator to generate JT files during Save or Save All operations. The latter operations will wait for the translator to complete its work, so "translate" can noticeably increase the time required to save models to Teamcenter. If the action is "none", the integration does not generate JT files.

Note

The translator starts a second Creo® session in the background. This will affect memory and license usage on the client workstation. It is recommended to install and configure the ProeToJT translator as part of the Teamcenter Dispatcher Services, so that JT translation occurs on a server-class computer and does not affect Creo® designers.

The action may also be "create_scripts", which causes the Integration to create all of the scripts and data files that would be created for the "translate" action, but the script that translate_script defines is not executed. The "create_scripts" action is intended for users who want to defer translation to a later time and who have a means for importing JT files into Teamcenter independently of the Integration. For example, a user may want to queue the translation script for overnight execution, instead of waiting for the translator during a save operation (the create_script tag could be used to define a command that will queue the script).

The save attribute indicates whether JT files should be saved to Teamcenter. If save is "true" and a JT file is found with the same name as a model being saved to Teamcenter, the JT file will be saved to Teamcenter with the model. If save is "false", JT files will not be saved to Teamcenter. The default is "true".

The eai_dir attribute specifies the path to the directory where the EAI Creo® translator is installed. This is not used by the Integration, but if defined, its value can be substituted into scripts you create (see the **JT-specific Substitution Keywords** section).

The *optional elements* are the additional tags described below.

command_file

```
<command_file [template="path"]/>
```

The `command_file` tag defines the translator command file, which lists every model to be translated and its corresponding configuration file. The `template` attribute specifies the path to a text file that serves as the template for each line in the command file.

Note that the command file template is really a template for a single line in the command file. When the `jt_file action` attribute is `"translate"`, the Integration creates a command file in the user's temp directory, and then it appends a copy of the template to the command file for each model that is to be translated.

The template may include any keyword described later in the **Substitution Keywords** section. The path to the template may include most of these keywords, except for those keywords that imply a single model. The default path to the template is `{ipem_dir}\pejt\proetojt.txt`.

configuration_file

```
<configuration_file [template="path"] [assembly_template="path"]/>
```

The `configuration_file` tag defines the translator configuration file. The `template` attribute specifies the path to a text file that serves as the template for the file. For each Creo® model to be translated, the integration copies this template to the user's temp directory and edits the copy to make substitutions for certain keywords. Since a configuration file is created for each model saved to Teamcenter, the path to the template and the template itself may include any of the keywords described later in the **JT-specific Substitution Keywords** section. The default path to the template is `{ipem_dir}\pejt\proetojt.config`.

If another template is specified with the `assembly_template` attribute, that template is used when the model is an assembly. If not specified, the assembly template is the same template used for all other models.

create_script

```
<create_script [template="path"] [cmd="command"]/>
```

The `create_script` tag defines a script that is created, and an operating system command that is executed, when the action is `"create_scripts"`. For example, the script might contain commands to run the translation script, then import JT files into Teamcenter, while the operating system command might be used to queue the script for overnight execution.

The `template` attribute specifies the path to a text file that serves as the template for the script. When the `jt_file action` attribute is `"create_scripts"`, the integration copies this template to the user's temp directory and edits the copy to make substitutions for certain keywords. The script is not executed unless the `cmd` attribute explicitly runs it. This script is created only once during a **Save** or **Save All** operation.

The path to the template, and the template itself, may include keywords described later in the Substitution Keywords section, except for those keywords that imply a single model. There is no default template.

The `cmd` attribute specifies an optional operating system command to execute when the action is `"create_scripts"`. This command string may also include the same keywords that can be used in the template. The command is executed only once during a **Save** or **Save All** operation, and only after

all scripts and data files have been created. There is no default command, but if the user chooses to define one, the integration will wait for it to complete before continuing the save operation, so commands should be implemented to run quickly.

file_name

```
<file_name pattern="pattern"/>
```

The `file_name` tag identifies the JT file by defining a pattern that matches the file's name. The same considerations that apply to the `file_name` tag for a general `auxiliary_file` definition also apply to `file_name` when used within a `jt_file` definition.

The `file_name` tag is optional. If it is not included in a `jt_file` definition, the Integration assumes each JT file is created in the same directory where its Creo® model is located.

The default configuration file sets the JT translator option *`structureOption`* to *`"FULL_SHATTER"`*, which produces a JT file in the same directory as the model. The *`file_name`* pattern that would match a file with the same name as the model, in the model's directory, is *`"{cad_name,lower}.jt"`*. Other *`structureOption`* settings may put the JT file elsewhere. For example, if *`structureOption`* is *`"PER_PART"`*, the JT file may be created in a subdirectory of the model's directory. A typical *`file_name`* pattern for this case would be *`"{cad_name,lower}/{cad_name,lower}.jt"`*.

Although part and assembly JT files might be placed in different directories as a result of using two different configuration templates in the *`configuration_file`* tag, only the files created by one of the templates can be matched by the *`file_name`* pattern. However, an *`auxiliary_file`* definition may be created in this situation to match the JT files created by the other template. For example, to match an assembly's JT file in a subdirectory of the model's directory:

```
<auxiliary_file cad_type="ASM">
       <pdm_location named_ref="JTPART" pdm_type="DirectModel"
            relation_type="IMAN_Rendering"/>
       <file_name pattern="{cad_name,lower}/{cad_name,lower}.jt"/>
    </auxiliary_file>
```

translate_script

```
<translate_script [template="path"] [cmd="command"] [ignore_status="true"|"false"]/>
```

The `translate_script` tag defines a script that, when executed by the operating system, runs the JT batch translator.

The `template` attribute specifies the path to a text file that serves as the template for the script. When the `jt_file action` attribute is `"translate"`, the Integration copies this template to the user's temp directory, edits the copy to make substitutions for certain keywords, then tells the operating system to execute the script. This happens only once during a **Save** or **Save As** operation, so the script must be capable of generating all requested JT files in one run.

The path to the template, and the template itself, may include keywords described later in the section, except for those keywords that imply a single model. The default path to the template is `{ipem_dir}\pejt\proetojt.bat`.

The `cmd` attribute specifies the operating system command that executes the translation script. This command string may also include the same keywords that can be used in the template. The default command is `{eai_translate_script}`.

If the operating system command exits with a non-zero status, the Integration usually assumes the command has failed and will not save the JT files. However, if `ignore_status="true"` is specified, the Integration ignores exit status and assumes the command always succeeds when executed.

JT-specific Substitution Keywords

The paths, command strings, and template files in the `jt_file` tag and its sub-elements support the same substitution keywords that the `file_name` pattern and `os_command` strings allow for general auxiliary files. Those keywords, which appear in patterns of the form {*keyword*}, are described previously in the **General Substitution Keywords** section.

In addition, the following keywords are also available when working with a `jt_file` tag and its sub-elements:

`{eai_command_file}`
: The absolute path to the EAI translator command file. The Integration creates this script from the `command_file` template.

`{eai_config_file}`
: The absolute path to a model's EAI translator configuration file. The Integration creates this script from the `configuration_file` template.

`{eai_create_script}`
: The absolute path to the script generated from the `create_script` template.

`{eai_dir}`
: The absolute path to the directory where the **PLM Components JT Translator for Creo®** software is installed. This is the path specified by the `eai_dir` attribute in the `jt_file` tag. If that attribute is not specified, this keyword will not be replaced.

`{eai_temp_dir}`
: The absolute path to a directory where the scripts, configuration files, and command files will be placed when created by the Integration.

`{eai_translate_script}`
: The absolute path to the script that, when executed, generates the JT files. The Integration creates this script from the `translate_script` template.

`{eai_instance_name}`
: If the model is a family table instance, this is the same as `{cad_name}`; otherwise its value is `GENERIC`. This is meant to be used as the value of the Instance option in the proConfig section of an EAI translator configuration file.

create_text_file

```
<create_text_file file_name="path" [separate_files="true"|"false"]
[header="path"] [body="path"] [footer="path"]
[phase="in_cad"|"in_directory"|"in_pdm"]/>
```

The `create_text_file` tag specifies one or more text files to create. By default, the file(s) will be created after Creo® has saved the models to the local file system, but before the models or auxiliary files are checked-in to Teamcenter.

Note

If an error occurs while attempting to write a text file, the file will not be checked into Teamcenter.

The file to be created is specified by the `file_name` attribute, where *path* is the path to the file. The path may be defined using keywords described in the section on **General Substitution Keywords**.

Note:

Literal paths, whether absolute or relative, are not supported, and will result in errors during **Save** to Teamcenter.

If `separate_files` is **true**, a separate text file will be created for each model saved to Teamcenter. If `separate_files` is **false**, a single file will be created, which may contain information about all models saved to Teamcenter.

Each text file is composed from three templates: the header, the body, and the footer. The header and footer appear once in the output file, at its beginning and end, respectively. The body appears between these two sections. If a single file is being created for all models, the body is repeated for each model being saved to Teamcenter. The body appears only once in the output file if a separate output file is created for each model.

Each template is optional, although at least one must be specified. The `header` attribute specifies the path to a text file that will be used as the header template, the `body` attribute specifies the path to a text file that will be used as the body template, and the `footer` attribute specifies the path to a text file that will be used as the footer template. The paths to these files may include keywords from the **General Substitution Keywords** section, and the text within these files may also use these keywords. If `separate_files` is true, keywords that imply a particular model, such as `{cad_name}` and `{cad_type}`, can be used in any template and in any file path. If `separate_files` is false, keywords that imply a particular model can be used only in the body template and in the path to the body template file. The header and footer, and the `file_name` path, can use only keywords that do not refer to a specific model when `separate_files` is false.

The use of these keywords in the paths to the template files makes it possible to use separate body templates for parts, assemblies, and drawings, or to use separate templates for family table instances and generics. For example, suppose the `{ipem_dir}` directory contains a header template file named `header.txt` with the following contents:

```
These models are being checked-in to Teamcenter…
```

and two body template files named `prt.txt` and `asm.txt`, in which `prt.txt` contains:

```
A part named {cad_name} is being checked-in to Teamcenter.
```

and `asm.txt` contains:

```
An assembly named {cad_name} is being checked-in to Teamcenter.
```

A `create_text_file` tag that uses these templates to create a single text file summarizing the parts and assemblies selected for check-in to Teamcenter might look like this:

```
<auxiliary_file cad_type="PRT:ASM">
      <create_text_file file_name="{temp_dir}\summary.txt"
      header="{ipem_dir}\header.txt"
      body="{ipem_dir}\{cad_type,lower}.txt"/>
</auxiliary_file>
```

When an assembly named COUPLING and two parts, PIN and SHAFT, are saved to Teamcenter, the contents of `summary.txt` would be:

```
These models are being checked-in to Teamcenter…
A part named PIN is being checked-in to Teamcenter.
A part named SHAFT is being checked-in to Teamcenter.
An assembly named COUPLING is being checked-in to Teamcenter.
```

The `{cad_type,lower}` pattern in the path to the body template would have been replaced with `prt` or `asm`, depending on the model type, and the `{cad_name}` pattern in the body templates would have been replaced by the model name.

You can also specify a phase in which the text file(s) will be created. Refer to the **os_command** tag for more information concerning the `phase` attribute. By default, text files are created during the `in_directory` phase.

Siemens Industry Software

Headquarters

Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

Americas

Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

Europe

Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

Asia-Pacific

Suites 4301-4302, 43/F
AIA Kowloon Tower, Landmark East
100 How Ming Street
Kwun Tong, Kowloon
Hong Kong
+852 2230 3308

About Siemens PLM Software

Siemens PLM Software, a business unit of the Siemens Industry Automation Division, is a leading global provider of product lifecycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.

© 2014 Siemens Product Lifecycle Management Software Inc. Siemens and the Siemens logo are registered trademarks of Siemens AG. D-Cubed, Femap, Geolus, GO PLM, I-deas, Insight, JT, NX, Parasolid, Solid Edge, Teamcenter, Tecnomatix and Velocity Series are trademarks or registered trademarks of Siemens Product Lifecycle Management Software Inc. or its subsidiaries in the United States and in other countries. All other trademarks, registered trademarks or service marks belong to their respective holders.